JavaScript (ES5 and earlier)

LAB GUIDE

THE JAVASCRIPT LANGUAGE (25 minutes)

1. Who created JavaScript and in which year? Which browser included it first? (*2 minutes*)

2. Find out answers to the following questions regarding JavaScript. Also compare it against a language like Java. (*23 minutes*)
   a. Is JavaScript an interpreted or compiled language?
   b. Is JavaScript a case-sensitive language?
   c. If you declare a number value and try to access the length property (which exists for string and arrays), what is the value of that expression?
   d. Will a code trying to call a function that has not been defined allowed to execute in the browser? If it is allowed, what happens when such code executes? Will the code after the call to the function that does not exist execute?

      ```
      foo(); // function does not exist
      console.log( 'code after call to foo' ); // will this line execute?
      ```

   e. Read about try..catch in JavaScript and modify the code snippet above so that the call to a non-existent function like foo logs an error message, yet continues executing code after it.

      Reference: https://javascript.info/try-catch
   f. How is JavaScript code included within an HTML page? How is a separate JavaScript file created and included in an HTML page?
   g. Which is the best place to include a script tag in an HTML file? Why?
      i) In the head
      ii) At the beginning of the body
      iii) At the end of the body (after rest of HTML elements within the body, but just before end of body)

CHROME DEVELOPER TOOLS AND HTTP (35 minutes)

1. Create an HTML page with a div/section with 3 paragraphs and some CSS applied to them both in inline fashion, i.e. using style attribute, and using an embedded style tag/external CSS (say color: olive). Open the Chrome Developer tools. (*10 minutes*)
   a. How can you check the HTML element hierarchy in it?
   b. How will you inspect the styles applied to any element, in particular a paragraph? How will you temporarily add CSS styles to the paragraph?
   c. Declare 2 variables in the console tab. In the same place, add them and store the result in another variable. Again in the console itself, log the result. Try to execute some more JavaScript in the console tab.

2. Open the Sources tab. Visit a website like YouTube or Medium (medium.com). Do you see the downloaded files? Use Ctrl+P (Windows) / Cmd+P (Mac OSX) to search and select a file. Open an HTML page and a JS file and check you can do so. (*4 minutes*)

3. Keep the Network tab open and repeat the website above. Which type of resource (called assets in website terminology) gets downloaded first - HTML/CSS/JavaScript? What assets are downloaded after the first asset downloads? Find the total number of files downloaded. What is the sum total of sizes of all the files (in MB). How much time did it take to load all the assets? (*6 minutes*)

4. Check the details of some network requests that went out in the previous exercise. (*15 minutes*)
   a. Check the HTTP Method used (GET/POST/... etc.). When is a GET request used. When does a POST request go out?
      Hint: For POST request check what happens in the network tab when you try to log in to some website - form data is usually POSTed.
   b. Continuing from the previous exercise, check out and note down some of the HTTP request headers and HTTP response headers. What goes out in the request body when data is POSTed? Find out another way to send data to the server apart from request body.
      Hint: Google for query parameters and how the query parameters are constructed in the URL. For example, go to a website like google.com or amazon.com and search for something in the search bar - you will see your search keywords go out in the request along with the URL as query parameters.

   Further exploration: The URL does not allow certain characters - for example spaces are not allowed. Find out what happens when you have multiple search keywords separated by spaces - How are these included in the query string? Find out what *URL encoding* is.

VARIABLES AND DATA TYPES (30 minutes)

1. Which of the following variable names are valid? (*2 minutes*)
   a. first name
   b. firstName
   c. 1stName
   d. _firstName
   e. $firstName

2. Suppose **`priceOfIPhone`** is not a variable declared with the var statement? Can we start using it like so? If we can, what is the scope of such a variable? (*4 minutes*)
   ```
   priceOfIPhone = 699;
   ```

3. Is a semi-colon necessary at the end of every statement? (*1 minute*)

4. Is there a separate data type for floating-point values in JavaScript or is it just number? (*1 minute*)

5. Declare variables that hold the name of a phone, price and availability at a store (like Best Buy). Choose appropriate data types for the values and assign the variables. Print the data types as well as the values of each. Print the number of characters in the name of the phone. (*6 minutes*)

6. Declare a variable but do not initialize it with a value. What is its value? Declare an object using object literal syntax and assign it to a variable. Log the variable. Later assign it the null value and log it again. (*8 minutes*)

7. What is the value of **`typeof null`**? What is the data type of the null value (be careful when concluding the data type). (*4 minutes*)

8. Classify the following as primitive and non-primitive. (*2 minutes*)
   a. 123
   b. "hello world"
   c. false
   d. { x: 100 }
   e. [1, 2, 3]
   f. [{ x: 1 }, { x : 2 }, { x :3 }]
   g. undefined
   h. null
   i. function foo() { }

9. What is printed in the console? Make sure you understand why the variable assumes the values that it does at different points in time. (*2 minutes*)

```
console.log( x );
var x = 100;
console.log( x );
```

OPERATORS (5 minutes)

1. What do the following statements print in the console? Do any of the statements throw an error? What is the relation between the first 4 statements and the last 4? (*4 minutes*)

```
console.log( 1 == 1 );
console.log( 1 === 1 );
console.log( 1 == '1' );
console.log( 1 === '1' );
console.log( 1 != 1 );
console.log( 1 !== 1 );
console.log( 1 != '1' );
console.log( 1 !== '1' );
```

2. What does the following expression evaluate to? What would the result in Java be? (*1 minute*)

```
1 / 2
```

ARRAYS

1. Create an array with a few numbers (at least 4). Now,
   a. Print the length of the array (number of items in the array)
   b. Increment the value of the first item
   c. Re-assign the value of the last item to the sum of the last 2 items.
   d. Add another number after the end of the array

   Example: If the array is [1, 3, 5, 7] and you need to add 9, it is added to the end of the array, and the array becomes [1, 3, 5, 7, 9]
   e. Declare a new array that is empty (has no items). Use a for loop to copy the array items into the new array

CONDITIONAL AND LOOP STATEMENTS (30 minutes)

1. Calculate the monthly cost of electricity used for a household. The billing is done as per following rules. The range includes the lower limit but excludes the upper limit.

Units	Cost per unit (in cents)
0 - 50	20
50 - 100	25
100 - 200	30
200 and above	35

If a household consumes 150 units of electricity, it will be billed at the rate of 20 cents for the first 50 units, 25 cents for the next 50 and 30 cents for the final 50 units. Thus the total cost would be 50 * 20 + 50 * 25 + 50 * 30 cents = $37.50

Create a variable **`units`** that holds the number of units consumed in a month in some household. Calculate the total cost of electricity consumed as per above rules and print it. (*16 minutes*)

2. Use a while loop to print the first 10 numbers in the following sequence (difference between consecutive numbers in the series keeps incrementing) (*10 minutes*)
   1, 2, 4, 7, 11, 16, ...

3. Repeat the above exercise with a for loop. (*4 minutes*)

4. Use a while loop to print the first 10 numbers in the following sequence
   1, 2, 4, 8, 16, 32, 64, ..., 2^(n - 1), ...

5. Use a for loop to print the first 10 numbers in the following sequence
   1, 2, 4, 8, 16, 32, 64, ..., 2^(n - 1), ...

6. Use for loops to print the following
   1
   1 2
   1 2 3
   1 2 3 4
   1 2 3 4 5
   Hint: Since console.log() prints a newline character at the end you will need to concatenate values before printing a line.

7. Use for loops to print the following
   1
   2 3
   4 5 6
   7 8 9 10
   11 12 13 14 15

8. Write a for loop that calculates sum of items in an array of numbers.
   Example: For array [1, 2, 3, 4] it calculates the sum as 10

9. Write a for loop that calculates sum of squares of items in an array of numbers.
   Example: For array [1, 2, 3, 4] it calculates the sum of squares as 30 (i.e. 1 + 4 + 9 + 16)

10. Write a for loop that creates a new array with squares of numbers in a supplied array.
    Example: Supplied array is [1, 2, 3, 4]. You should generate a new array [1, 4, 9, 16] from it.

11. The following are train stations (as they appear) when you take the VTA train between San Francisco and San Jose. (*35 minutes*)

```
const stations = [ "San Francisco", "22nd Street", "Bayshore", "San Bruno",
"Millbrae Transit Center", "Burlingame", "San Mateo", "Hayward Park",
"Hillsdale", "Belmont", "San Carlos", "Redwood City", "Menlo Park", "Palo
Alto", "California Avenue", "San Antonio", "Mountain View", "Sunnyvale",
"Lawrence", "Santa Clara", "College Park", "San Jose" ];
```

    The regions are divided into 4 Zones. Once you cross over to a new Zone, you remain in that Zone till you cross over to an entirely new Zone. You never enter or exit the same Zone more than once. The Zones are listed here - http://www.caltrain.com/stations/systemmap.html

    The first station in each Zone is given in the following array.

```
const firstStations = [ "San Francisco", "Millbrae Transit Center", "Menlo
Park", "Lawrence" ]
```

    a. Add a new station 'Tamien' at the end of stations array
    b. Change the second station name to "Twenty Second Street"
    c. Print the number of stations

d. By iterating through the 2 arrays, create a new array (say firstStationsIndexes) with indexes of the firstStations. The result should be

```
firstStationsIndexes; // [ 0, 4, 13, 19 ]
```

e. Declare a new array that is empty (has no items). Use a for loop to copy the stations array items into the new array. How is this different from creating a reference to the array like so?
const stationCopy = stations;

Hint: During assignments, primitives (for example - strings) are copied by value, non-primitives (for example - arrays) are copied by reference.

f. Generate an array where every item corresponds to a station, and has the station's zonal code instead of the station's name. Thus, the output array should be [1, 1, 1, 1, 2, 2, ...] corresponding to ["San Francisco", "22nd Street", "Bayshore", "San Bruno", "Millbrae Transit Center", ...]

FUNCTIONS

1. Write a function square() that returns the square of a number passed to it. Use function declaration syntax to declare the function.
***Example*:**
```
console.log( square( 3 ) ); // prints 9
```

2. Write a function that logs "Good morning", "Good afternoon", "Good evening", or "Good night" based on the hour of the day, and call it.
    - 5 <= hour < 12: "Good morning"
    - 12 <= hour < 16: "Good afternoon"
    - 16 <= hour < 20: "Good evening"
    - Otherwise, "Good night"

Note: The hour can be obtained by using this statement
```
const hour = (new Date()).getHours();
```

3. Write a function *sumArray* that calculates and returns the sum of items in an array of numbers that is passed to it, and call it like so
```
const result = sumArray( [ 1, 2, 3, 4 ] );
console.log( result ); // prints 10
```

4. Write a function *sumSquares* that creates a new array with squares of numbers in the array passed to it, and call it like so
```
const result = sumSquares( [ 1, 2, 3, 4 ] );
console.log( result ); // prints 30
```

5. Write a function *contains* that accepts an array, and a number and returns true if the number appears in the array, and false otherwise. Use function expression syntax to declare the function
```
console.log( contains( [ 1, 2, 3, 4 ], 3 ) ); // prints true
console.log( contains( [ 1, 2, 3, 4 ], 5 ) ); // prints false
```

6. Write a function that accepts another function and calls the accepted function

7. Write a function *sum* that accepts 2 numbers (say *x* and *y*) and another function (say, *transform*) as arguments. The transform function should be a function that accepts a number and returns another number - for example, a function *square* that accepts a number and returns the square of a number. The sum() function applies the transform function on each of each and y and returns the sum of the results of calling transform - for example, sum() would return $x^2 + y^2$ if called as sum(x, y, square);

   Example:

```
function square( x ) { return x * x };
function cube( x ) { return x * x * x };

console.log( sum( 2, 3, square ) ); // prints 13
console.log( sum( 2, 3, cube ) ); // prints 35
```

8. Write a function *sumArray* that works like so.

```
console.log( sumArray( [ 1, 2, 3 ], square ) ); // prints 14
console.log( sumArray( [ 1, 2, 3 ], cube ) ); // prints 36
```

9. Write a function *exponentFactory* that accepts a number, say x. Define 2 functions *square* and *cube* within it (which accept a number each, and return the square and cube respectively). If x is 2, exponentFactory returns the square function, if 3 it returns the cube function. For any other input it returns a function that returns the number it accepts as such. Call the exponentFactory() function and then the returned function, and log the result.

   Example:

```
let fn;

fn = exponentFactory( 2 );
console.log( fn( 5 ) ); // prints 25;

fn = exponentFactory( 3 );
console.log( fn( 5 ) ); // prints 125;

fn = exponentFactory( 4 );
console.log( fn( 5 ) ); // prints 5;
```

10. Write a function *addTo* that accepts a number x. addTo() returns a function that accepts a number y and returns the sum of x and y. Call the addTo() function few times, and then the returned function, and log the result.

    Example:

```
const addTo10 = addTo( 10 );
console.log( addTo10( 5 ) ); // prints 15
console.log( addTo10( 7 ) ); // prints 17

const addTo20 = addTo( 20 );
console.log( addTo20( 5 ) ); // prints 25
console.log( addTo20( 7 ) ); // prints 27
```

OBJECTS (2 hours)

1. Create 2 objects (that represents 2 persons, say John and Jane), each with 2 properties - name (a string), and age (a number). (*20 minutes*)
   - Print John's age.
   - Increase Jane's age and print the Jane object.
   - Add an address property to John and set it to an object with "first line" and "city" as properties (the values for these properties also need to be set).
   - Print John's city name
   - Add a new property spouse to each object. Set John's spouse property to Jane object, and Jane's spouse property to John object
   - Add an emailids property to Jane. Set it to an array with 2 strings representing Jane's email ids.
   - Print the second email id of Jane.
   - Change the second email id of Jane and print it.
   - Add a third email id for Jane and print the Jane object.
   - Add a method celebrateBirthday() on John that adds 1 to the John's age. Call it on John to increase John's age.
   - Add a method celebrateBirthday() on Jane that adds 1 to the Jane's age. Call it on Jane to increase Jane's age.
   - Wouldn't it be nice to have a single celebrateBirthday() method shared by both John and Jane objects? Declare celebrateBirthday() as a global function and set it up as a method on both John and Jane objects. Call it to check it increases the age.

2. Create a movie object that represents details of your favorite movie. Suggested information to have in the object - name, cast (an array of strings with cast member's names), yearOfRelease, boxOfficeCollection, addToCast(newMember) that accepts a new cast member's name and adds to the cast array, addToCollection(amount) that accepts box office collections for a week and adds it to the current boxOfficeCollection. (*10 minutes*)

3. Use object literal syntax to create an object that represents details of a product sold on an e-commerce store. (*45 minutes*)
   Initial set of details would be
   - id - a unique alphanumeric id for the product (string)
   - name (say iPhone 11)
   - manufacturer (say Apple)
   - releaseDate - a JavaScript Date object with release date of phone
   - price (say, 699) and currency (say, USD) - these 2 details need to be grouped together as an object
   - starRating (1 - 5)
   - specs (needs to be grouped together) - an object that includes color (string), memory (object or string), front and rear Camera resolutions (object with resolution of front and rear cameras (string/number))
   - available colors - an array of all colors (string) the phone is available in

   Reference: For how to create a Date object, refer https://javascript.info/date

   After creating the object, do the following.
   - Print the phone's full name (manufacturer followed by name - eg. Apple iPhone 11)
   - Increase the price of the phone by $50

- Add an availableSpecs property which is an array of objects. Each object represents the specs for a variant of the phone.
- Add a dimensions property which is an object with height, width, depth.
- Print the resolutions of the front and rear cameras separately
- Create 2 objects for 2 other variants of the iPhone and add a similarProducts property on each of the 3 objects (the original object and the 2 new ones). Each should be set to an array with items being the other 2 phones. Now choose any one phone object, and do the following.
- Print the memory size of the second in the list of similarProducts
- Create an object for one more phone variant and add it to the array of variants.
- Use a for loop to print the names of the variants of the phone
- Define a ```calculateDiscoutedPrice``` method that returns the discounted price when passed a discount percentage. When an invalid percentage is passed, it should throw an error. Make sure to put the call to the method within a try..catch, and handle the error by logging it.
- Wouldn't it be nice to have a single calculateDiscoutedPrice method shared by both all phone object? Declare calculateDiscoutedPrice as a global function and set it up as a method on each of the phone objects. Call it to check it returns the discounted price.

4. Create an object representing booking for a train ticket. (*45 minutes*)
Details would include passenger name, age, source and destination, train number (unique for every train), name of train, train timings (departure from source and arrival at destination), price of ticket, and class (First Class, General). Choose appropriate data types, and group related information as sub-objects (for example details of passenger could be grouped as an object within the ticket object).

   Define another object ```bookings``` that represents bookings for a few trains (as an object with ```reserve``` and ```cancel``` methods, and array of objects called ```trainBookings```). Each object in the array represents the booking details for a train - it would have the train number (unique for every train), and tickets available in the 2 classes - First Class and general. Define a ```reserve``` method that when passed the train number, number of seats and class, checks the availability and deducts the specified number of seats if available - it should generate a unique booking id and return it - the booking id is unique over all bookings for all trains - you can use a global variable for this. The booking details are also maintained in the object representing the train's bookings (as an array with booking details). If required number of seats are not available it simply returns false. Also define a ```cancel``` method that accepts the unique booking id and searches through all bookings and removes the booking details. It also adds the number of seats in the booking back to the number of available seats for the booked class.

FUNCTION CONTEXT

1. Declare a function *foo* and log its context
   - Use bind() to create a new function where the context is the object { x: 1 } instead
   - Call the bound function

2. Declare a function *sum* that accepts 2 numbers *x* and *y*, and returns their sum.
   - Use bind() to create a new function where x is bound to 10, and the context is unchanged. Call the bound function with a value for y, and log the result.
   - Use bind() to create a new function where x is bound to 10, and y to 20 (context is unchanged). Call the bound function, and log the result.

BUILT-IN CLASSES – STRINGS, ARRAYS

1. Given the following snippet of code, solve the questions that follow.

```
const numbers = [ 5, 11, 13, 7, 2, 31, 3, 19, 23, 17, 29 ];
```

   - Sort the numbers in the array in increasing order and print the array
   - Sort the array in decreasing order and print the array
   - Add the number 37 to the end of the array using push()
   - Remove the last 2 numbers in the array
   - Remove the numbers at indices 3, 4 (i.e. the 4th and 5th numbers) and insert the strings 'Seven' and 'Eleven' in their place.
   - Use indexOf() to check if 23 belongs to the array or not. Also, check if 41 belongs to the array or not.

2. Given the following array, solve the questions that follow using appropriate array iterator methods (forEach, find, filter, map)

```
   const days = [ 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday' ];
```

   - Print the name of each item in the array
   - Create a new array with the number of letters in each day (example, Sunday has 6 letters). Thus the new array that should be generated would be [6, 6, 7, 9, 8, 6, 8]
   - Create a new array with the days that begin with letters S - Z. Thus the new array that should be generated would be ['Sunday', 'Tuesday', 'Wednesday', 'Thursday', 'Saturday']
   - Create a new array with days that have exactly 6 letters. Thus the new array that should be generated would be ['Sunday', 'Monday', 'Friday']

3. Given the following array, solve the questions that follow using appropriate array iterator methods (forEach, find, filter, map)

```
   const phones = [
      { name : 'Samsung Galaxy S10+ Plus', price: 620, type: 'refurbished',
manufacturer: 'Samsung' },
      { name : 'Apple iPhone 7 Plus', price: 450, type: 'refurbished',
manufacturer: 'Apple' },
      { name : 'One Plus 6', price: 430, type: 'new', manufacturer: 'OnePlus' },
      { name : 'Apple iPhone Xs', price: 910, type: 'new', manufacturer: 'Apple'
},
      { name : 'One Plus 7', price: 430, type: 'refurbished', manufacturer:
'OnePlus' },
      { name : 'Apple iPhone 8 Plus', price: 610, type: 'new', manufacturer:
    'Apple' },
    ];
```

- Create a new array with the name of each phone. Thus the new array that should be generated would be ['Samsung Galaxy S10+ Plus', 'Apple iPhone 7 Plus', ...]
- Set a new price for every phone by adding a tax of 5%
- Create a new array with details of only the new phones
- Find the first phone whose price is less than $500 and print it

4. Given the following array, solve the questions that follow using appropriate array iterator methods (forEach, find, filter, map, reduce, some)

```
const persons = [
    {
        name: 'John',
        salary: 1000,
        age: 32,
        emails: [
            'john@gmail.com',
            'john@example.com'
        ]
    },
    {
        name: 'Jane',
        age: 28,
        salary: 2000,
        emails: [
            'jane@gmail.com',
            'jane@example.com',
            'jane@yahoo.com',
        ]
    },
    {
        name: 'Mark',
        age: 46,
        salary: 3000,
        emails: [
            'mark@gmail.com',
            'mark@example.com'
        ]
    },
    {
        name: 'Mary',
        age: 44,
        salary: 4000,
        emails: [
            'mary@gmail.com',
            'mary@example.com'
        ]
    }
];
```

- Find all persons whose are 35 years or older
- Find a person who has at least 3 email ids
- Find the sum of salaries for all persons
- Get a list of users along with their first email id. The result should be an array like this. Do not modify the given array.

```
[
   {
```

```
        name: 'John',
        email: 'john@gmail.com'
    },
    {
        name: 'Jane',
        email: 'jane@gmail.com'
    },
    {
        name: 'Mark',
        email: 'mark@gmail.com'
    },
    {
        name: 'Mary',
        email: 'mary@gmail.com'
    }
];
```

- Find a person who has a yahoo.com email id (Hint: you can use the **some** array iterator method along with string / regular expression methods to test an email)

5. Copy the array from https://workshops-server.herokuapp.com/workshops in a variable. Solve the questions that follow using appropriate array iterator methods (forEach, find, filter, map, reduce)
   - Find all **frontend** workshops
   - Find all workshops in **Bangalore** city
   - Find the first workshop available online. How will you find the last workshop available online (one way is to use findLast, but this method is not supported on many mobile browsers)
   - Find the first workshop available in both online and in-person modes
   - Get the number of online workshops
   - Get a list of the imageUrls
   - Get a list of workshops with the following structure

```
[
    { name: 'Angular JS Bootcamp', id: 1 },
    { name: 'React JS Masterclass', id: 2 }
    ...
]
```

   - Get a single string which has the names of frontend workshops separated by commas. The result should look like so (use **filter** along with **join**)

```
'Angular JS Bootcamp, React JS Masterclass, …'
```

6. Given the following string, solve the questions that follow.

```
let quote = '  With great power comes great responsibility.  ';
```

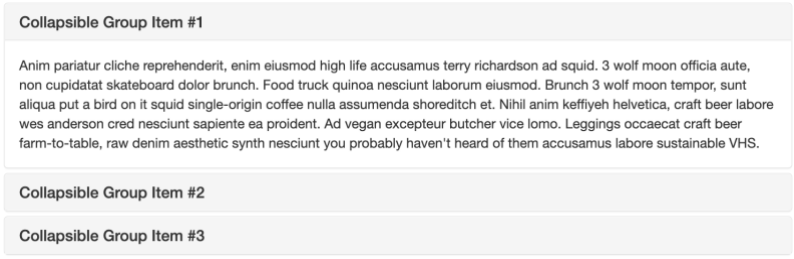
   - Create a string from the given string (quote) by replacing 'responsibility' with 'electricity bill'
   - Print the index of the first occurence of the word 'great'
   - Print the first 10 letters of the string
   - Remove the leading and trailing spaces
   - Form a string with the letters in upper case
   - Form an array with all the words in the quote as items in the array

DOM AND EVENT HANDLING

1. Create an HTML page with 3 paragraphs (give each a unique id). Now do the following by manipulating the DOM via JavaScript.
   - Create a new paragraph with some text within, and add it before the first paragraph.
   - Create a new paragraph with some text within, and add it to the end of the document body
   - Print the HTML within the second paragraph to the console
   - Replace the contents of the second paragraph with 2 spans, each with some text
   - Print the id of the second paragraph
   - Change the id of the second paragraph
   - Set the name attribute for the second paragraph to some value. Use setAttribute() to do so.
   - Set styles for the third paragraph and turn it to crimson color background with white colored text
   - Define a class in CSS that sets a black border, small padding and olive colored text. Add the class to the last paragraph.

2. Create an HTML page with a paragraph, a form with text input (with minlength set to 8) and a submit button. Now do the following.
   - Set up an event handler that shows an alert dialog with text of the paragraph when it is clicked
   - Set up an event handler that turns the background color of the input to lightgreen when the number of characters within the input is 8 or more (this happens as the user types)
   - Set up an event handler that prevents form submission when the submit button is clicked. It then checks to see if number of characters within the input is at least 8. If so, it submits the forms. Else, an appropriate error message is displayed below the input.

3. Create a Bootstrap-like panel component with a header and body. Give it appropriate styles using CSS.
   Reference: https://getbootstrap.com/docs/3.4/javascript/#collapse-example-accordion



   - Set up an event handler so that the panel body toggles (opens/closes) when the header is clicked
   - Add more such panels to your page and make sure clicking a panel's header toggles only that panel's body
   - Create a function that accepts a panel DOM node and turns it into a toggleable panel

4. Create a page with 2 inputs (both accepting number), button, and a span (initially empty).
   - When the button is clicked, the two numbers within the input are added and displayed inside the span

- Now add a dropdown (select input) with options +, -, *, / (default selection is +). When user select an option, the contents of the span change to reflect the result of the new operation.

AJAX

1. Within an HTML page, write a script to fetch data about things to do from https://jsonplaceholder.typicode.com/todos. Display the results in a list in the page. Make sure to display an error message in the same page in case of any errors.

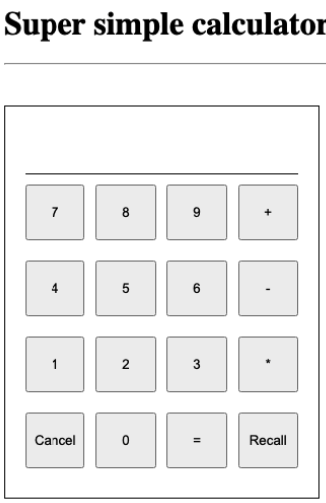
2. Within an HTML page, write a script to fetch data about products from https://awesome-store-server.herokuapp.com/products. Display the results in a card layout (gallery). The image of the product should be placed on top every card. Make sure to display an error message in the same page in case of any errors.

OBJECT PROTOTYPE

1. Create an object vehicle with name (string), averageSpeed (number) and type (string with value "air" | "water" | "land"). Create 2 other objects - car, aeroplane, with vehicle as their prototype. Add engineCapacity, typeOfFuel to each and set them. To the car object, add numAirbags. To the aeroplane add thrust.

CALCULATOR IMPLEMENTATION IN JAVASCRIPT

Define a calculator like the following



Instructions

1. Define an IIFE to avoid creating global variables.
2. Define an array to hold results, and an index to maintain for last recalled result (lastRecalled) initialized to 0.
3. Define a function onNumberPress() that is invoked on click of number buttons (".btn-number"). It reads the number (text within the button), and appends it to the display input's (".display") value.
4. Define a function onOperatorPress() that is invoked on click of operator buttons (".btn-op"). It reads the operator (text within the button), and appends it to the display input's value. But before doing so it checks if the last character displayed is an operator (an operator cannot be followed by another operator).
5. Define a function onCancelPress() that is invoked on click of cancel button (".btn-cancel"). It clears the display and resets lastRecalled to last index of results array (so that a future recall button press will start recall with latest results).
6. Define a function onEvalPress() that is invoked on click of evaluate button, i.e. the = button (".btn-eval"). It reads the display value and evaluates it using JavaScript's eval() function, and shows the result in the display. It also saves the result to the results array. Set lastRecalled to last index of results array.
7. Define a function onRecallPress() that is invoked on click of recall button (".btn-recall"). It decrements lastRecalled value, gets the value of results at lastRecalled index and sets the display to that value.