CodeQuotient

Department of Computer Science & Engineering

LAB MANUAL

22CS014-Front-End Engineering-II - 3rdSem - 2023

Student Name	ANUJ KUMAR
Email	anuj1699.be23@chitkara.edu.in
Course Name	22CS014-Front-End Engineering-II - 3rdSem - 2023

Faculty Incharge

Table of Contents

Aim: JS: The console.log method

The console is a panel that displays important messages, like errors, for developers. In javascript, the console is an object which provides access to the browser debugging console. It’s going to be very useful for us to print values to the console, so we can see the work that we’re doing.
console.log("CodeQuotient");
This example logs CodeQuotient to the console. The semicolon denotes the end of the line, or statement. Although in JavaScript your code will usually run as intended without a semicolon, we recommend learning the habit of ending each statement with a semicolon so you never leave one out in the few instances when they are required.

Task 1: Print "Hello CodeQuotient!" using console.log method.

Solution:

```
/* Type your javascript code here */
console.log("Hello CodeQuotient!");
```

Aim: JS: Data types

A value in JavaScript is always of a certain type. For example, a string or a number.
There are eight basic data types in JavaScript. We can put any type in a variable. For example, a variable can at one moment be a string and then store a number:
// no error
let message = "hello";
message = 123456;
Programming languages that allow such things, such as JavaScript, are called “dynamically typed”, meaning that there exist data types, but variables are not bound to any of them.

1. Number
let n = 123;
n = 12.345;
The number type represents both integer and floating point numbers.
There are many operations for numbers, e.g. multiplication *, division /, addition +, subtraction -, and so on.
Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: Infinity, -Infinity and NaN.
Infinity represents the mathematical Infinity " . It is a special value that’s greater than any number.
We can get it as a result of division by zero:
console.log(1 / 0); // Infinity
NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:
console.log("not a number" / 2); // NaN, such division is erroneous
NaN is sticky. Any further operation on NaN returns NaN:
console.log("not a number" / 2 + 5); // NaN

2. String
A string in JavaScript must be surrounded by quotes.
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
In JavaScript, there are 3 types of quotes.
Double quotes: "Hello".
Single quotes: 'Hello'.
Backticks: `Hello`.
Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.
Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:
let name = "John";
// embed a variable
alert(`Hello, ${name}!`); // Hello, John!
// embed an expression
alert(`the result is ${1 + 2}`); // the result is 3

3. Boolean (logical type)
The boolean type has only two values: true and false.
This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.

For instance:
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
Boolean values also come as a result of comparisons:
let isGreater = 4 > 1;
alert(isGreater); // true (the comparison result is "yes")

4. null
The special null value does not belong to any of the types described above.
It forms a separate type of its own which contains only the null value:
let age = null;
In JavaScript, null is not a "reference to a non-existing object" or a "null pointer" like in some other languages.
It's just a special value which represents "nothing", "empty" or "value unknown".
The code above states that age is unknown.

5. undefined
The special value undefined also stands apart. It makes a type of its own, just like null.
The meaning of undefined is "value is not assigned".
If a variable is declared, but not assigned, then its value is undefined:
let age;
alert(age); // shows "undefined"
Technically, it is possible to explicitly assign undefined to a variable:

6. object
The object type is special.
All other types are called "primitive" because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.
Being that important, objects deserve a special treatment.

Task 1: Initialize variable named 'num' with a number.
Task 2: Initialize variable named 'str' with a string.
Task 3: Initialize variable named 'bool' with a boolean.

Solution:

let num = 42;
let str = "Hello, World!";
let bool = true;

ANUJ KUMAR
anuj1699.be23@chitkara.edu.in

Aim: JS: Arithmetic Operators

An operator is a character that performs a task in our code. JavaScript has several built- in arithmetic operators, that allow us to perform mathematical calculations on numbers.
Addition +,Subtraction -,Multiplication *,Division /,Remainder %
The first four are straightforward, while the remainder operator %, despite its appearance, is not related to percentages.
The result of a % b is the remainder of the integer division of a by b.

Task 1: Print the result of following statements (**Hint:** use console.log method):
a + ba - ba * ba / ba % b

Note: a and b are already declared in the scope.

Solution:

```
/* Type your javascript code here */
// Addition
console.log(a + b);
// Subtraction
console.log(a - b);
// Multiplication
console.log(a * b);
// Division
console.log(a / b);
// Remainder
console.log(a % b);
```

Aim: JS: String Concatenation

Let’s meet features of JavaScript operators that are beyond school arithmetic.
Usually, the plus operator + sums numbers.
But, if the binary + is applied to strings, it merges (concatenates) them:
let s = "my" + "string";
alert(s); // mystring
Note that if any of the operands ialert(6 - '2'); // 4, converts '2' to a number
alert('6' / '2'); // 3, converts both operands to numberss a string, then the other one is converted to a string too.
For example:
alert('1' + 2); // "12"
alert(2 + '1'); // "21"
See, it doesn’t matter whether the first operand is a string or the second one.
Here’s a more complex example:
alert(2 + 2 + '1'); // "41" and not "221"
Here, operators work one after another. The first + sums two numbers, so it returns 4, then the next + adds the string 1 to it, so it’s like 4 + '1' = 41.
The binary + is the only operator that supports strings in such a way. Other arithmetic operators work only with numbers and always convert their operands to numbers.
Here’s the demo for subtraction and division:
alert(6 - '2'); // 4, converts '2' to a number
alert('6' / '2'); // 3, converts both operands to numbers

Task 1: Print the result of following statements (**Hint:** use console.log method):
str1 + str2str1 - numstr2 * num

Note: *str1*, *str2* and *num* are already declared in the scope.

Solution:

```
console.log(str1 + str2);
console.log(str1 - num);
console.log(str2 * num);
```

Aim: JS: Variables

let :-
To create a variable in JavaScript, use the let keyword.
The statement below creates (in other words: declares) a variable with the name “message”:
let message;
Now, we can put some data into it by using the assignment operator =:
let message;

message = 'Hello'; // store the string
The string is now saved into the memory area associated with the variable. We can access it using the variable name:
let message;
message = 'Hello!';
alert(message); // shows the variable content
To be concise, we can combine the variable declaration and assignment into a single line:
let message = 'Hello!'; // define the variable and assign the value
alert(message); // Hello!
We can also declare multiple variables in one line:
let user = 'John', age = 25, message = 'Hello';

There are some rules while declaring a JavaScript variable (also known as identifiers).
Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign. After first letter we can use digits (0 to 9), for example value1.JavaScript variables are case sensitive, for example x and X are different variables.

Var :-
The var declaration is similar to let. Most of the time we can replace let by var or vice-versa and expect things to work:
var message = "Hi";
alert(message); // Hi
But internally var is a very different beast, that originates from very old times. It’s generally not used in modern scripts, but still lurks in the old ones.
If you don’t plan on meeting such scripts you may even skip this chapter or postpone it.
On the other hand, it’s important to understand differences when migrating old scripts from var to let, to avoid odd errors.
Correct JavaScript variables:
var x = 10;
var _value="sonoo";
Incorrect JavaScript variables:
var 123=30;
var *aa=320;

const :-
To declare a constant (unchanging) variable, use const instead of let:
const myBirthday = '15.04.1997';
Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:
const myBirthday = '15.04.1997';
myBirthday = '15.06.1999'; // error, can't reassign the constant!
When a programmer is sure that a variable will never change, they can declare it with const to guarantee and clearly communicate that fact to everyone.

Task 1: Initialize a let variable named message with "Hello CodeQuotient!".
Task 2: Initialize a var variable named number with 999.
Task 3: Initialize a const variable named SINE_90 with 1.

Solution:

```
let message = "Hello CodeQuotient!";
var number = 999;
const SINE_90 = 1;
```

Aim: JS: Assignment Operator

The simple assignment operator (=) is used to assign a value to a variable. The assignment operation evaluates to the assigned value. Chaining the assignment operator is possible in order to assign a single value to multiple variables

```
let x = 2;
const y = 3;

console.log(x);
// expected output: 2

console.log(x = y + 1); // 3 + 1
// expected output: 4

console.log(x = x * y); // 4 * 3
// expected output: 12

// Assuming the following variables
 // x = 5
// y = 10
// z = 25

x = y // x is 10
x = y = z // x, y and z are all 25
```

Task 1: declare a variable sum and assign a value of a + b.

Note: a and b are already declared in global scope.

Solution:

```
let sum = a + b;
```

Aim: JS: Conditional Statements

Conditional Statements :-
Sometimes, we need to perform different actions based on different conditions.
To do that, we can use the if statement and the conditional operator ?, that's also called a "question mark" operator.
if :-
The if(...) statement evaluates a condition in parentheses and, if the result is true, executes a block of code.

```
let x = 6;
if (x == 6) console.log(`x = ${x}`);
```

The if (…) statement evaluates the expression in its parentheses and converts the result to a boolean.
Let's recall the conversion rules from the chapter Type Conversions:
A number 0, an empty string "", null, undefined, and NaN all become false. Because of that they are called "falsy" values.
Other values become true, so they are called "truthy".
So, the code under this condition would never execute:

```
if (0) { // 0 is falsy
 ...
}
```

…and inside this condition – it always will:

```
if (1) { // 1 is truthy
 ...
}
```

We can also pass a pre-evaluated boolean value to if, like this:

```
let cond = (x == 6); // equality evaluates to true or false
if (cond) {
 ...
}
```

else :-
The if statement may contain an optional "else" block. It executes when the condition is false.
For example:

```
let x="CodeQuotient"

if (x == "CodeQuotient") {
 console.log("Get Better At Programming");
} else {
 console.log(`x is not equal to ${x}`); // any value except CodeQuotient
}
```

Comparison Operators :-
Comparison operators are used in logical statements to determine equality or difference between variables or values.
Given that x = 10, the table below explains the comparison operators:
less than: <

```
x<8 // false
x<12 // true
```

greater than: >

```
x>8 // true
x>12 // false
```

less than or equal to: <=

x<=8 // false
x<=12 // true
x<=10 // true
greater than or equal to: >=
x>=8 // true
x>=12 // false
x>=10 // true
Is equal to: ===
X === 10 // true
X === “10” // false
Is not equal to: !==
X !== 10 // false
X !== “10” // true

Logical Operators :-
Logical operators are used to determine the logic between variables or values.
Given that x = 6 and y = 3, the table below explains the logical operators:
and: &&
(x < 10 && y > 1) is true
or: ||
(x == 5 || y == 5) is false
Not: !
!(x == y) is true

Ternary Operator :-
Sometimes, we need to assign a variable depending on a condition.
For instance:

```
let accessAllowed;
let age = prompt('How old are you?', '');

if (age > 18) {
 accessAllowed = true;
} else {
 accessAllowed = false;
}
alert(accessAllowed);
```

The so-called “conditional” or “question mark” operator lets us do that in a shorter and simpler way.
The operator is represented by a question mark ?. Sometimes it’s called “ternary”, because the operator has three operands. It is actually the one and only operator in JavaScript which has that many.
The syntax is:
let result = condition ? value1 : value2;
The condition is evaluated: if it’s truthy then value1 is returned, otherwise – value2.

else-if ladder :-
Sometimes, we’d like to test several variants of a condition. The else if clause lets us do that. The else if statements allow you to have multiple possible outcomes. if/else if/else statements are read from top to bottom, so the first condition that evaluates to true from the top to bottom is the block that gets executed.

```
if (condition)
  statement 1;
```

```
else if (condition)
  statement 2;
.
.
else
  statement;
```

if-else-if ladder helps users decide from among multiple options. The "if" statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the else-if ladder is bypassed.

Task 1: Write a program to find, if the input number is even.
Input Format: *An integer.*
Output Format: *Boolean. Assign the result to variable named isEven.*

Solution:

```
function isEven(input) {
    let isEven = (input % 2 === 0); // Check if the number is even
    return isEven;
}
```

ANUJ KUMAR
anuj1699.be23@chitkara.edu.in

Aim: JS: The switch keyword

else-if statements are a great tool if we need to check multiple conditions. In programming, we often find ourselves needing to check multiple values and handling each of them differently

```
let grade = 'B';

switch (groceryItem) {
 case 'A':
  console.log('A Grade');
  break;
 case 'B':
  console.log('B Grade');
  break;
 case 'C':
  console.log('C Grade');
  break;
 default:
  console.log('Fail');
  break;
}
```

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.
The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Task 1: A user wants to have a software which print different messages for each number from 1 to 5, and a message saying "wrong number entered!" if anything else is provided as an input. below are the list of messages to print for valid inputs.
1: 'Hi there!'2: 'How are you?'3: 'I hope you're doing great!'4: 'How may I help you?'5: 'Thanks for visiting us.'
Input Format: *A single integer.*
Output Format: *string. Assign the result to variable named message.*

Solution:

```
function printMessage(input) {
    let message;
    switch (input) {
        case 1:
            message = 'Hi there!';
            break;
        case 2:
            message = 'How are you?';
            break;
        case 3:
            message = "I hope you're doing great!";
            break;
        case 4:
            message = 'How may I help you?';
            break;
        case 5:
            message = 'Thanks for visiting us.';
            break;
        default:
            message = 'Wrong number entered!';
            break;
    }
    return message;
}
```

Aim: JS: Loops

Loops in programming come into use when we need to repeatedly execute a block of statements until a specific condition is met.
We often need to repeat actions.
For example, outputting goods from a list one after another or just running the same code for each number from 1 to 10.
Loops are a way to repeat the same code multiple times.Loops allow us to create efficient code that automates processes to make scalable, manageable programs.

The “while” loop :-

The while loop starts by evaluating the condition. If the condition is true, the statement(s) is/are executed. If the condition is false, the statement(s) is/are not executed. After that, while loop ends.
Here is the syntax for while loop:

```
while (condition) {
  // so-called "loop body"
}
```

For instance, the loop below outputs i while i <= 3:

```
let i = 1;
while (i <= 3) { // shows 1, then 2, then 3
 alert( i );
 i++;
}
```

A single execution of the loop body is called an iteration. The loop in the example above makes three iterations. If i++ was missing from the example above, the loop would repeat (in theory) forever. In practice, the browser provides ways to stop such loops, and in server-side JavaScript, we can kill the process.
Any expression or variable can be a loop condition, not just comparisons: the condition is evaluated and converted to a boolean by while.

The “do…while” loop :-

The condition check can be moved below the loop body using the do..while syntax:

```
do {
 // loop body
} while (condition);
```

The loop will first execute the body, then check the condition, and, while it’s truthy, execute it again and again.
For example:

```
let i = 1;
do {
 console.log( i );
 i++;
} while (i <= 3);
```

This form of syntax should only be used when you want the body of the loop to execute at least once regardless of the condition being truthy.

The “for” loop :-

The for loop is the most commonly used loop.
It looks like this:

```
for (begin; condition; step) {
 // ... loop body ...
}
```

Let’s learn the meaning of these parts by example. The loop below runs alert(i) for i from 0 up to (but not including) 3:

```
for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
 alert(i);
}
```

In for loop, begin executes once, and then it iterates: after each condition test, body and step are executed.
If you are new to loops, it could help to go back to the example and reproduce how it runs step-by-step on a piece of paper.
Here's exactly what happens in our case:

```
// for (let i = 0; i < 3; i++) alert(i)
// run begin
let i = 0
// if condition !' run body and run step
if (i < 3) { alert(i); i++ }
// if condition !' run body and run step
if (i < 3) { alert(i); i++ }
// if condition !' run body and run step
if (i < 3) { alert(i); i++ }
// ...finish, because now i == 3
```

"break" statement :-
Normally, a loop exits when its condition becomes falsy.
But we can force the exit at any time using the special break directive.
When you use break,it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.

```
for (var x = 1; x < 10; x+=2) {
  if (x === 7) {
    break;
  }
  console.info('x is ' + x);
}
// output:
// x is 1
// x is 3
// x is 5
```

"continue" Statement :-
The continue directive is a "lighter version" of break. It doesn't stop the whole loop. Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).
We can use it if we're done with the current iteration and would like to move on to the next one.
The loop below uses continue to output only odd values:

```
for (var x = 1; x < 10; x+=2) {
  if (x === 7) {
    continue;
  }
  console.info('x is ' + x);
}
// output:
// x is 1
// x is 3
// x is 5
// x is 9
```

Nested Loops :-

When we have a loop running inside another loop, we call that a nested loop.
For example, in the code below we loop over i and j, prompting for the coordinates (i, j) from (0,0) to (2,2):

```
for (let i = 0; i < 3; i++) {
 for (let j = 0; j < 3; j++) {
  console.log (`${i},${j})`);
 }
}
```

Task 1: Write a program to find the count of prime numbers in a series from 2 to n. [2, n]
Input Format: *A single integer (n). 2 <= n <= 100*
Output Format: *number. Assign the result to variable named primeCount.*

Solution:

```
function primeCount(n) {
    let primeCount = 0;
    function isPrime(num) {
        if (num <= 1) return false;
        for (let i = 2; i <= Math.sqrt(num); i++) {
            if (num % i === 0) return false;
        }
        return true;
    }
    for (let i = 2; i <= n; i++) {
        if (isPrime(i)) {
            primeCount++;
        }
    }
    return primeCount;
}
```

Aim: JS: Arrays

array is a single variable that is used to store different elements. It is often used when we want to store a list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements.
There are two syntaxes for creating an empty array:
let arr = new Array();
let arr = [];
Almost all the time, the second syntax is used. We can supply initial elements in the brackets:
let friends = ["Abhi", "Sumit", "Pawan"];
Array elements are numbered, starting with zero.
We can get an element by its number in square brackets:
let friends = ["Abhi", "Sumit", "Pawan"];
alert(friends[0]); // Abhi
alert(friends[1]); // Sumit
alert(friends[2]); // Pawan
We can replace an element:
friends[2] = 'Pranav'; // now ["Abhi", "Sumit", "Pranav"]
…Or add a new one to the array:
friends[3] = 'Lemon'; // now ["Abhi", "Sumit", "Pranav", "Lemon"]
The total count of the elements in the array is its length:
let friends = ["Abhi", "Sumit", "Pawan"];
alert(friends.length); // 3
We can also use alert to show the whole array.
let friends = ["Abhi", "Sumit", "Pawan"];
alert(friends); // Abhi,Sumit,Pawan
You may recall that you can declare variables with both the let and const keywords. Variables declared with let can be reassigned.
Variables declared with the const keyword cannot be reassigned. However, elements in an array declared with const remain mutable. Meaning that we can change the contents of a const array, but cannot reassign a new array or a different value.
Also, When an array contains another array it is known as a nested array.
let arrNested = ["A" , ["B1" , "B2"] , ["C1" , "C2"]];
arrNested[1]; // ["B1" , "B2"]
arrNested[2][0] // C1

Task 1: Write a program to find the length of an array.
Input Format: An array of size n.
Output Format: *size of the input array. Assign the array length to variable named length.*

Solution:

```
function getArrayLength(input) {
    // Assign the length of the array to the variable 'length'
    let length = input.length;
    return length;
}
```

Aim: JS: Arrays Iterators - map

.map() is called on an array, it takes an argument of a callback function and returns a new array! Take a look at an example of calling .map():

```
let numbers = [1, 2, 3, 5, 7, 11];

// pass a function to map
const mapArray = numbers.map(function(x){ return x*2; })

console.log(mapArray); // expected output: [2, 4, 6, 10, 14, 22]
```

Task 1: Write a program to shallow copy an array.
Input Format: An array of size n.
Output Format: Copied array of size n.

Solution:

```
function shallowCopy(input) {
    // Check if the input is an array
    if (!Array.isArray(input)) {
        throw new Error("Input must be an array");
    }
    // Use the slice method to create a shallow copy
    return input.slice();
}
```

Aim: JS: Arrays Iterators - forEach

forEach() calls a provided callback function once for each element in an array in ascending order. It is not invoked for index properties that have been deleted or are uninitialized.

```
let numbers = [2, 3, 5, 7, 9, 11];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
 console.log(`${arr[index]} = ${value}`)
}
```

callback is invoked with three arguments:
the value of the elementthe index of the elementthe Array object being traversed
The range of elements processed by forEach() is set before the first invocation of callback. Elements which are appended to the array after the call to forEach() begins will not be visited by callback. If elements that are already visited are removed (e.g. using shift()) during the iteration, later elements will be skipped.The return value for .forEach() will always be undefined.

Task 1: Write a program to add 1 to each element of input array.
Input Format: An array of size n.

Solution:

```
function addOneToEach(input) {
  // Iterate through each element of the array
  for (let i = 0; i < input.length; i++) {
    // Add 1 to the current element
    input[i] += 1;
  }
  // Return the modified array
  return input;
}
```

Aim: JS: Arrays Iterators - filter

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

```
let ageArray = [45, 4, 9, 16, 25];
let over18 = ageArray.filter(function (age) {
 return age > 18;
});
Console.log ( over18 ) // [ 45, 25 ]
```

Age > 18; is the condition in the callback function. Any age from the ageArray that’s value is greater than 18 will be added to the over18 array.

Task 1: Write a program to remove all even numbers from an array.
Input Format: An array of size n.
Output Format: Return an array containing only odd numbers. If none return an empty array.

Solution:

```
function removeEven(input) {
  // Filter the array to remove even numbers
  return input.filter(num => num % 2 !== 0);
}
```

Aim: JS: Object Methods

An object in JavaScript may also have a function as a member, in which case it will be known as a method of that object.
Let us see such an example :

```
let school = {
  name: 'CodeQuotient ',
  location : 'Mohali',
  established : 2015,
  displayInfo : function(){
    console.log(`${school.name} was established
       in ${school.established} at ${school.location}`);
  }
}
school.displayInfo();
```

In the above example, “displayinfo” is a method of the school object that is being used to work with the object’s data, stored in its properties.
“this” in methods :-
It’s common that an object method needs to access the information stored in the object to do its job.
For instance, the code inside school.displayInfo() may need the name of the school.
To access the object, a method can use this keyword.
The value of this is the object “before dot”, the one used to call the method.

```
let school = {
  name: 'CodeQuotient ',
  location : 'Mohali',
  established : 2015,
  displayInfo : function(){
    console.log(`${this.name} was established
       in ${this.established} at ${this.location}`);
  }
}
```

Task 1: Implement below mentioned methods on the given object named ladder.
setSteps :- takes no. of steps as a parameter, and set the steps accordingly.**getSteps :-** return the no. of steps.**stepUp :-** Increase the no. of steps by 1.**stepDown :-** Decrease the no. of steps by 1.

Solution:

```
ladder.setSteps = function(numSteps) {
    this.steps = numSteps;
};
ladder.getSteps = function() {
    return this.steps;
};
ladder.stepUp = function() {
    this.steps += 1;
};
ladder.stepDown = function() {
    if (this.steps > 0) { // Ensure steps don't go below zero
```

```
            this.steps -= 1;
        }
    };
```

Aim: JS Assignment: Calculate Bill

You have been given 2 arrays of objects (menu and categories).
Below is the structure of **categories**:-

```
const categories = [{
  id : "C1",
  categoryName : "Platters",
  superCategory : {

    superCategoryName : "South Indian",
    id : "SC1"
  }
}, {...}];
```

Below is the structure of **menu**:-

```
const menu = [{ id : "item1",
    itemName : "Butter Roti",
    rate : 20,
    taxes : [
      {
        name : "Service Charge",
        rate : 10,
        isInPercent : false
      }, {
        name : "GST",
        rate : 5,
        isInPercent : true
      }
    ],
    category : {
      categoryId : "C2"
    }

  }, {...}];
```

A variable **bill** is being passed into the function **calculateBill()**.
bill has the following structure:-

```
bill = { id : "B1",
  billNumber : 1,
  opentime : "06 Nov 2020 14:19",
  customerName : "CodeQuotient",
  billItems : [
    {
      id : "item2",
      quantity : 3,
      discount : {
        rate : 10,
        isInPercent : true
      }

    },
    {
      id : "item3",
      quantity : 5,
      discount : {
        rate : 10,
        isInPercent : false
      }

    }, {...},
  ]
};
```

Task: Complete the given function **calculateBill()** to calculate the total amount of the order. It also stores the bill items information in variable billItems in the format given below.

Bill Item Info Format: <item name>@<item price per piece> x <item ordered quantity> = <total item amount>
Input format: function parameter bill will be passed a bill object value.
Output format: calculateBill() returns an array having two values as total bill amount and bill items array respectively.

NOTES:
1. The menu & categories variables are already defined as per the above given definition.
2. Round of all the final prices to 2 decimal places. e.g. - 6.606666 will be rounded to 6.61
3. Taxes and charges will be calculated on the discounted price, i.e. calculate the discount before calculating the taxes and charges.
4. If 'isInPercent' is set to true, that means that particular discount/tax/charge will be calculated as the percentage of the base price or discounted price (e.g. 10% discount on 100 rupees would be 10 rupees). Otherwise it would be absolute (e.g. 5 rupee discount on 100 rupees).

Sample Testcase with Explanation:-
Sample Input:-

```
{
 "id":"B1",
 "billNumber":10,
 "openTime":"12/01/2021, 10:43:39",
 "customerName":"CodeQuotient",
 "billItems":[
   {
    "id":"item2",
    "quantity":3,
    "discount":{
      "rate":10,
      "isInPercent":false
    }
   },
   {
    "id":"item1",
    "quantity":15,
    "discount":{
      "rate":10,
      "isInPercent":true
    }
   },
   {
    "id":"item4",
    "quantity":2,
    "discount":{
      "rate":20,
      "isInPercent":false
    }
   },
   {
    "id":"item3",
    "quantity":5,
    "discount":{
      "rate":5,
      "isInPercent":true
    }
   }
 ]
}
```

Sample Output:-

```
1391.00
Paneer Butter Masala@120 x 3 = 455.40
Butter Roti@20 x 15 = 345.60
Dosai Platter@150 x 2 = 286.00
Masala Dosai@50 x 5 = 304.00
```

Explanation:-

The first line of output is the total bill generated, i.e. 1391.00

1. **Item2** is Paneer Butter Masala and has a rate of 20 with service charge 10%, GST 18% & Service Tax of 10% & has a discount of 10.
 Without taxes the price of the item is 120. After discount it will be 120-10 = 110.
 Applying charges/taxes on it will be:
 Service Charge: 10% of 110 = 11 ,
 GST: 18% of 110 = 19.8
 Service Tax: 10% of 110 = 11.
 Price after Adding taxes on the item = 110 + 11 + 19.8 + 11 = 151.8
 Total Price of the product = Price of Single Item(After adding taxes) * quantity of the product = 151.8 * 3 = 455.40

2. **Item1** is Butter Roti and has a rate of 20 with service charge 10% & GST 18% & has a discount of 10%.
 Without taxes the price of the item is 20. After applying discount it will be 20 - (10% of 20) = 20-2 = 18
 Applying charges/taxes on it will be:
Service Charge: 10% of 18 = 1.8
 GST: 18% of 18 = 3.24
 Price after Adding Taxes on the item = 18 + 1.8 + 3.24 = 23.04

Total Price of the product = Price of Single Item (After adding taxes) * quantity of the product = 23.04 * 15 = 345.60

And similarly for other items....

Solution:

```
function calculateBill(bill) {
  let totalBillAmount = 0;
  let billItems = [];
  bill.billItems.forEach(billItem => {
    const menuItem = menu.find(item => item.id === billItem.id);
    // Calculate the discounted price
    let discountedPrice = menuItem.rate;
    if (billItem.discount.isInPercent) {
      discountedPrice -= (menuItem.rate * billItem.discount.rate / 100);
    } else {
      discountedPrice -= billItem.discount.rate;
    }
    // Calculate the total price with taxes
    let totalPrice = discountedPrice;
    menuItem.taxes.forEach(tax => {
      if (tax.isInPercent) {
        totalPrice += (discountedPrice * tax.rate / 100);
      } else {
        totalPrice += tax.rate;
      }
    });
    // Calculate the total item amount
    const totalItemAmount = totalPrice * billItem.quantity;
    // Update total bill amount
    totalBillAmount += totalItemAmount;
    // Format the bill item info
    const billItemInfo = `${menuItem.itemName}@${menuItem.rate} x ${billItem.quantity}
= ${totalItemAmount.toFixed(2)}`;
    billItems.push(billItemInfo);
  });
  // Round off total bill amount to 2 decimal places
  totalBillAmount = totalBillAmount.toFixed(2);
  return [totalBillAmount, billItems];
}
```