

Frontend With JS and React JS

A SUMMER INTERNSHIP

Submitted by

KEVISH NIRMAL THAKKAR

210160107505

In partial fulfilment for the award of the degree of

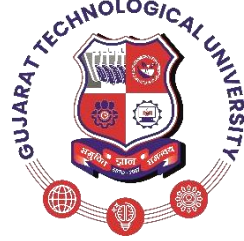
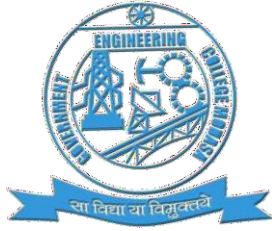
BACHELOR OF ENGINEERING

In

**COMPUTER ENGINEERING DEPARTMENT GOVERNMENT ENGINEERING
COLLEGE, MODASA**



**Gujarat Technological University, Ahmedabad August,
2023**



GOVERNMENT ENGINEERING COLLEGE

Shamlaji Road, Aravali District, Modasa, Gujarat 383315

CERTIFICATE

This is to certify that the project report submitted along with the project entitled
“Frontend With JS and React JS ” has been carried out by **Kevish Nirmal Thakkar**
(210160107505) under my guidance in partial fulfillment for the degree of Bachelor of Engineering
in Computer Engineering, 7th Semester of Gujarat Technological University, Ahmadabad during the
academic year 2022-23.

Prof. Nainesh Nagekar

Internal Guide

Prof. M.B Chaudhari

Head of the Department

(C.E./ I.T. Department)

Joining Letter



July 18, 2023,

Ahmedabad

Dear **Kevish Thakkar,**

We are pleased to offer you an internship at **Lanatus Systems LLP** (<https://lanatussystems.com/>) in the **Software Engineer** department at our **Ahmedabad** office. Your internship shall commence on **27 July 2023**.

The terms and conditions of your internship with the Company are set forth below:

1. Subject to your acceptance of the terms and conditions contained herein, your project and responsibilities during the Term will be determined by the supervisor assigned to you for the duration of the internship.
2. The duration of the internship is 15 Days.
3. Your timings will be Full-time (9.30 am to 6.30 pm), Monday to Friday.
4. Based on your performance you may receive an offer of full-time employment from us. The details of this will be discussed towards the end of your Internship.

Please reply to this mail with your intent to join the internship. That would be considered as your agreement to the above terms.

For, Lanatus Systems LLP

Authorized Signatory

AGREED TO & ACCEPTED

Signature



Completion Certificate



Date: August 7, 2023

Location: Ahmedabad

Certificate of Internship

I hereby certify that **Mr. Kevish Thakkar** student of Government Engineering College, Modasa. Completed the internship from 27 July 2023 to 10 August 2023 in Frontend (Javascript, ReactJS, and ReduxJS). I confirm that Kevish was able to satisfactorily deliver several features of the project.

Project Manager Information:

Name: Prakash Dudhat
Company: Lanatus Systems LLP
Email ID: prakashd@lanatussystems.com

During the internship, he demonstrated good technical skills with a self-motivated attitude to learn new things. His performance exceeded expectations and was able to make a significant contribution to the development of the project.

Best Regards,

A handwritten signature in black ink, appearing to read 'Maulik'.

Maulik Shah (C.E.O)





**Government Engineering College, Modasa
Shamlaji Road, Aravali District, Modasa Gujarat 383315.**

DECLARATION

I hereby declare that the Summer Internship report submitted along with the Summer Internship entitled “**Frontend with JS and React JS**” submitted in partial fulfilment for the degree of Bachelor of Engineering in Computer Engineering to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me at Lanatus Sysytem LLP . under the supervision of Prof. Nainesh Nagekar and that no part of this report has been directly copied from any students’ reports or taken from any other source, without providing due reference.

Name of student

Kevish N Thakkar

Acknowledgement

I would like to express my deepest gratitude to all those who provided me the possibility to the completion of the internship. A special gratitude of thanks I give to our assistant Professor, **Prof. Nainesh Nagekar**, whose contribution in stimulating suggestions and encouragement, helped me to coordinate the internship especially in drafting this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the Head of Department, **Prof. Minubhai Chaudhary**, who gave the permission to use all required equipment and the necessary material to fulfill the task. Last but not the least, many thanks go to the teachers and my friends and families who have invested their full effort in guiding us in achieving the goal.

Also I appreciate the guidance given by the developer at Lanatus, **Mr Maulik Shah** as well as the panels especially for the internship that has advised me and gave guidance at every moment of the internship.

Abstract

A front-end developer is a type of software developer who specializes in creating and designing the user interface (UI) and user experience (UX) of websites and web applications. The primary responsibility of a front-end developer is to ensure that the visual and interactive aspects of a website or application are user-friendly, aesthetically pleasing, and functionally efficient.

Front-end developers work with various technologies, tools, and languages, including:

1. **HTML (HyperText Markup Language):** The standard markup language used to create the structure and layout of web pages.
2. **CSS (Cascading Style Sheets):** A stylesheet language used to control the presentation, formatting, and appearance of web pages, such as colors, fonts, and layout.
3. **JavaScript:** A programming language that allows developers to add interactivity, animations, and other dynamic elements to websites and web applications.

Front-end developers may also use libraries and frameworks, such as React, Angular, or Vue.js, to streamline their work and create more sophisticated and interactive UIs. Additionally, they often collaborate with back-end developers, who are responsible for the server-side logic and data management, to ensure seamless integration between the front-end and back-end components of a web application or website.

<u>Index</u>		
<i>Sr No</i>	<i>Title</i>	<i>Sign</i>
<u>1</u>	<u>Day 1</u>	
<u>2</u>	<u>Day 2</u>	
<u>3</u>	<u>Day 3</u>	
<u>4</u>	<u>Day 4</u>	
<u>5</u>	<u>Day 5</u>	
<u>6</u>	<u>Day 6</u>	
<u>7</u>	<u>Day 7</u>	
<u>8</u>	<u>Day 8</u>	
<u>9</u>	<u>Day 9</u>	
<u>10</u>	<u>Day 10</u>	
<u>11</u>	<u>Day 11</u>	
<u>12</u>	<u>Day 12</u>	
<u>13</u>	<u>Day 13</u>	
<u>14</u>	<u>Day 14</u>	
<u>15</u>	<u>Day 15</u>	
<u>16</u>	<u>Conclusion</u>	
<u>17</u>	<u>Summary</u>	

- **DAY – 1**

- 1) What is Javascript and IDE?
 - 2) Hello World Program , Variables, Data types.
 - 3) Pop-box, Type Conversion, Operators.
-

➔What is Javascript?

It is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

When JavaScript was created, it initially had another name: “LiveScript”. But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

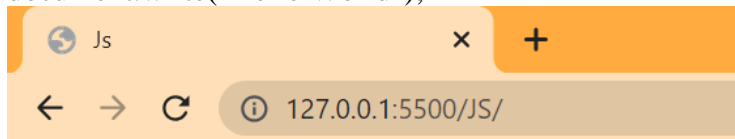
➔What is IDE?

The term IDE (Integrated Development Environment) refers to a powerful editor with many features that usually operates on a “whole project.” As the name suggests, it’s not just an editor, but a full-scale “development environment.”

An IDE loads the project (which can be many files), allows navigation between files, provides autocompletion based on the whole project (not just the open file), and integrates with a version management system (like git), a testing environment, and other “project-level” stuff.

➔Hello World Program.

```
document.write("Hello World");
```



Hello World

➔A Variable.

A variable is a "named storage" for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the `let`, `var` and `const` keyword.

Ex:- let message = "hello";

var a = 25;

const PI = 3.14;

➔Data Types:-

Javascript is dynamically typed scripting language that means we did not need to assign the type to data while declaring the variable. It will automatically define the type of the data as per the value stored in it.

Ex:- let message = "hello";

Document.write(typeof(message)); //o/p string

There are 8 basic data types in JavaScript.

- Seven primitive data types:
 - number for numbers of any kind: integer or floating-point, integers are limited by $\pm(2^{53}-1)$.
 - bigint for integer numbers of arbitrary length.
 - string for strings. A string may have zero or more characters, there's no separate single-character type.
 - boolean for true/false.
 - null for unknown values – a standalone type that has a single value null.
 - undefined for unassigned values – a standalone type that has a single value undefined.
 - symbol for unique identifiers.
- And one non-primitive data type:
 - object for more complex data structures.

The typeof operator allows us to see which type is stored in a variable.

- Usually used as `typeof x`, but `typeof(x)` is also possible.
- Returns a string with the name of the type, like "string".
- For `null` returns "object" – this is an error in the language, it's not actually an object.

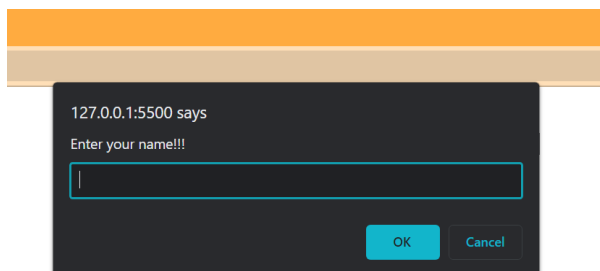
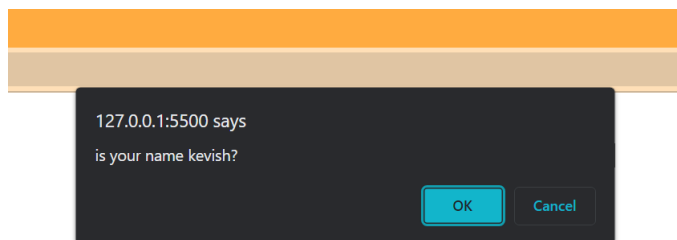
➔Pop-up Boxes

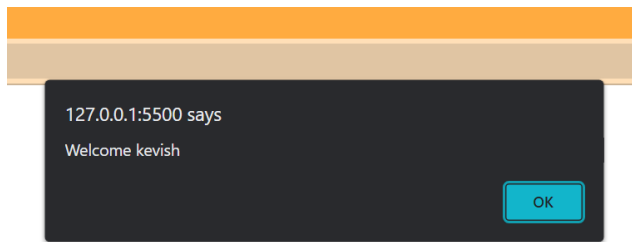
Javascript contains three popup boxes named i) confirm ii) alert iii) prompt.

Example:-

```
let name = prompt("Enter your name!!!");  
  
if(confirm(`is your name ${name}?`)){  
    alert(`Welcome ${name}`);  
}else{  
    alert(`Write your name`);  
}
```

Output:-





➔Type Conversion

➔Ex-1

```
let value = true;  
alert(typeof value); // boolean
```

```
value = String(value); // now value is a string "true"  
alert(typeof value); // string
```

➔Ex-2

```
let str = "123";  
alert(typeof str); // string
```

```
let num = Number(str); // becomes a number 123
```

```
alert(typeof num); // number
```

➔Ex-3

```
alert( Boolean(1) ); // true  
alert( Boolean(0) ); // false
```

➔Operators

➔Math Operator:-

The following math operations are supported:

- Addition +,
- Subtraction -,
- Multiplication *,
- Division /,
- Remainder %,
- Exponentiation **.

➔Increment/Decrement Operator

Increasing or decreasing a number by one is among the most common numerical operations.(++ and --)

The operators ++ and -- can be placed either before or after a variable.

When the operator goes after the variable, it is in “postfix form”: counter++.

The “prefix form” is when the operator goes before the variable: ++counter.

Both of these statements do the same thing: increase counter by 1.

→Example:- (Show the output)

```
"" + 1 + 0
"" - 1 + 0
true + false
6 / "3"
"2" * "3"
4 + 5 + "px"
"$" + 4 + 5
"4" - 2
"4px" - 2
" -9 " + 5
" -9 " - 5
null + 1
undefined + 1
" \t \n" - 2
```

Output:-

```
"" + 1 + 0 // "10"
"" - 1 + 0 //-1
true + false //true //1
6 / "3" //2
"2" * "3" //6
4 + 5 + "px" //9px
"$" + 4 + 5 //$45
"4" - 2 //2
"4px" - 2 //NaN
" -9 " + 5 // -9 5
" -9 " - 5 //-14
null + 1 //1
undefined + 1 //nan
" \t \n" - 2 //-2
```

- **DAY – 2**

1) Conditional Statements

2) Loops

➔ **Conditional Statements.**

➔ If Statement:-

The if(...) statement evaluates a condition in parentheses and, if the result is true, executes a block of code.

➔ Else Statement:-

The if statement may contain an optional else block. It executes when the condition is falsy.

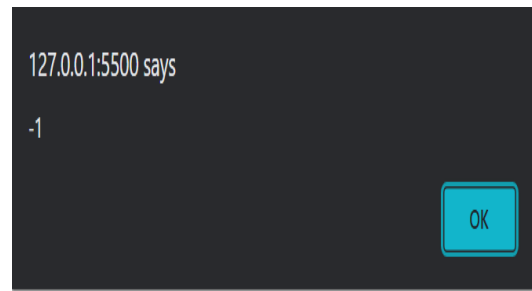
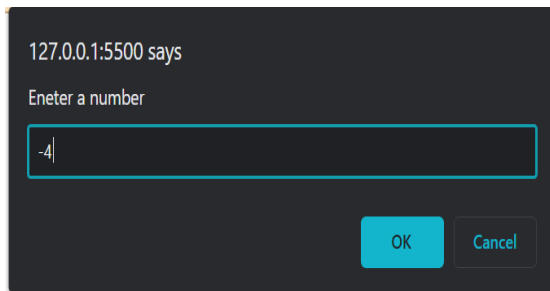
Example:- Using if..else, write the code which gets a number via prompt and then shows in alert:

- 1, if the value is greater than zero,
- -1, if less than zero,
- 0, if equals zero.

Code:-

```
let number = Number(prompt("Enter a number"));
if(number > 0){
    alert("1");
}else if(number < 0){
    alert("-1");
}else if(number === 0){
    alert("0");
}else{
    alert("Kindly provide number only!!!!");
}
```

Output:-



→Loops

→While Loop:-

The while loop has the following syntax:

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

While the condition is truthy, the code from the loop body is executed.

→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 Loop:-

The for loop is more complex, but it's also the most commonly used loop.

It looks like this:

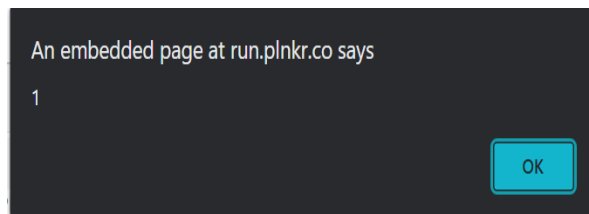
```
for (begin; condition; step) {  
  // ... loop body ...  
}
```

Example:-

1) `let i = 3;`

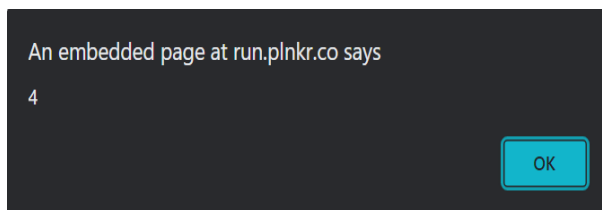
```
while (i) {  
  alert( i-- );  
}
```

Output:-



2) `for (let i = 0; i < 5; i++) alert(i);`

Output:-



3)Output Prime Numbers:-

```
let number = Number(prompt("Enter a number"));
```

```
for(let j = 2;j<=number;j++){
```

```
  let flag = true;
```

```
  for (let i = 2; i < j; i++) {
```

```
    console.log(j)
```

```
    if (j % i == 0) {
```

```
      flag = false;
```

```
      break;
```

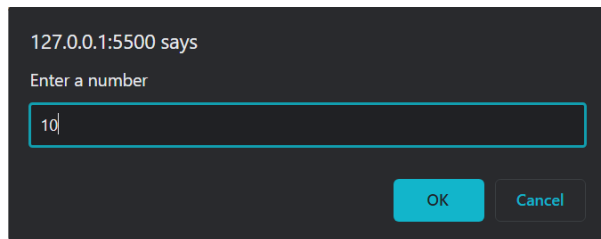
```
    }
```

```
  }
```

```
  if(flag){
```



```
document.write(j+", ")  
  
}  
  
}
```



Output:-



2, 3, 5, 7,

- **DAY – 3**

1) Switch Case

2) Functions

➔Switch Case

The switch has one or more case blocks and an optional default.

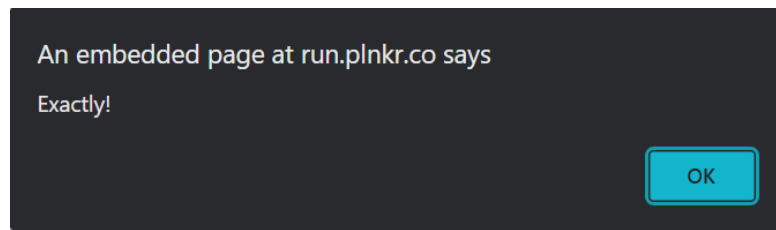
It looks like this:

```
switch(x) {  
  case 'value1': // if (x === 'value1')  
    ...  
    [break]  
  
  case 'value2': // if (x === 'value2')  
    ...  
    [break]  
  
  default:  
    ...  
    [break]  
}
```

Example:-

```
let a = 2 + 2;  
  
switch (a) {  
  case 3:  
    alert( 'Too small' );  
    break;  
  case 4:  
    alert( 'Exactly!' );  
    break;  
  case 5:  
    alert( 'Too big' );  
    break;  
  default:
```

Output:-



➔Functions

Quite often we need to perform a similar action in many places of the script.

For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.

Functions are the main “building blocks” of the program. They allow the code to be called many times without repetition.

We’ve already seen examples of built-in functions, like `alert(message)`, `prompt(message, default)` and `confirm(question)`. But we can create functions of our own as well.

➔Function Declaration:-

The function keyword goes first, then goes the *name of the function*, then a list of *parameters* between the parentheses (comma-separated, empty in the example above, we’ll see examples later) and finally the code of the function, also named “the function body”, between curly braces.

Syntax:- `function name(parameter1, parameter2, ... parameterN) {
 // body
}`

➔Parameters:-

When a value is passed as a function parameter, it’s also called an argument.

In other words, to put these terms straight:

A parameter is the variable listed inside the parentheses in the function declaration (it’s a declaration time term).

An argument is the value that is passed to the function when it is called (it’s a call time term).

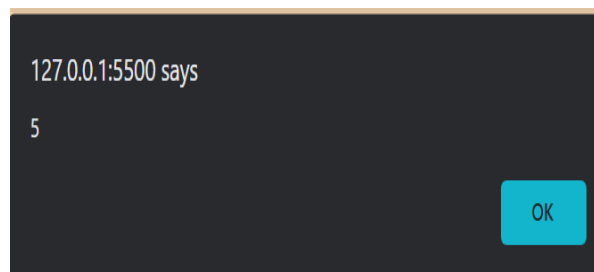
→Example:-

1) Minimum of 2.

```
function min(a,b){  
  if(a<b){  
    return a;  
  }return b;  
}
```

```
let minimum = min(5,10);  
alert(minimum)
```

Output:-

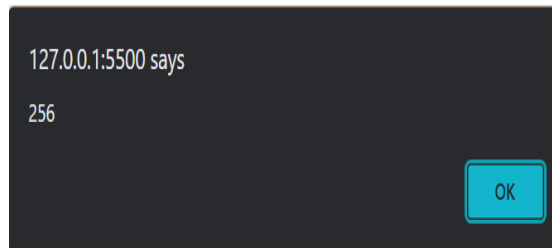


2) Power of n.

```
function pow(x,n){  
  let num =1;  
  while(n>0){  
    num *=x;  
    n--;  
  }  
  return num;  
}
```

```
let power = pow(2,8);  
alert(power)
```

Output:-



- **DAY – 4**

-
- 1) Objects
 - 2) Garbage Collection
 - 3) Numbers.
-

➔Objects

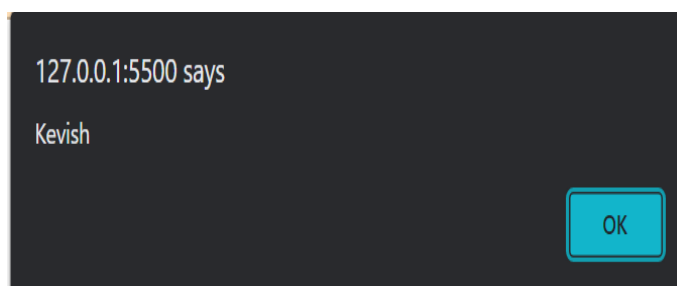
There are eight data types in JavaScript. Seven of them are called “primitive”, because their values contain only a single thing (be it a string or a number or whatever).

In contrast, objects are used to store keyed collections of various data and more complex entities. In JavaScript, objects penetrate almost every aspect of the language. So we must understand them first before going in-depth anywhere else.

An object can be created with figure brackets {...} with an optional list of properties. A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything.

Syntax:- `let user = new Object(); // "object constructor"`
 `let user = {};`

Example:- `let user = { // an object`
 `name: "Kevish", // by key "name" store value "Kevish"`
 `age: 21 // by key "age" store value 21`
 `}`
 `let key = "name";`
 `alert(user[key]);`



➔For..in Loop

To walk over all keys of an object, there exists a special form of the loop: for..in. This is a completely different thing from the for(;;) construct that we studied before.

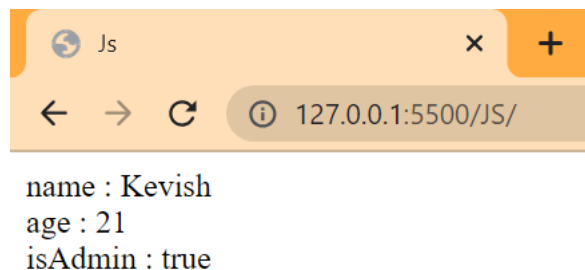
The syntax:

```
for (key in object) {  
  // executes the body for each key among object properties  
}
```

Example:-

```
let user = {  
  name: "Kevish",  
  age: 21,  
  isAdmin: true  
};  
  
for (let key in user) {  
  document.write(key + " : " + user[key] + "<br>")  
}
```

Output:-



➔Garbage Collection

As stated above, the general problem of automatically finding whether some memory "is not needed anymore" is undecidable. As a consequence, garbage collectors implement a restriction of a solution to the general problem. This section will explain the concepts that are necessary for understanding the main garbage collection algorithms and their respective limitations.

➔ Numbers Functions

For regular number tests:

`isNaN(value)` converts its argument to a number and then tests it for being NaN

`Number.isNaN(value)` checks whether its argument belongs to the number type, and if so, tests it for being NaN

`isFinite(value)` converts its argument to a number and then tests it for not being NaN/Infinity/-Infinity

`Number.isFinite(value)` checks whether its argument belongs to the number type, and if so, tests it for not being NaN/Infinity/-Infinity

For converting values like 12pt and 100px to a number:

Use `parseInt/parseFloat` for the “soft” conversion, which reads a number from a string and then returns the value they could read before the error.

For fractions:

Round using `Math.floor`, `Math.ceil`, `Math.trunc`, `Math.round` or `num.toFixed(precision)`.

Make sure to remember there's a loss of precision when working with fractions.

More mathematical functions:

See the `Math` object when you need them. The library is very small, but can cover basic needs.

➔Example:-

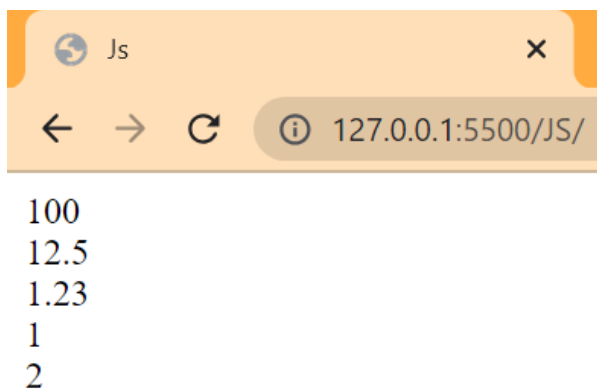
```
document.write(parseInt("100px"))
```

```
document.write("<br>")
```



```
document.write(parseFloat("12.5em"));
document.write("<br>")
let num = 1.23456;
document.write(Math.round(num * 100) / 100);
document.write("<br>")
document.write(Math.floor(num));
document.write("<br>")
document.write(Math.ceil(num));
```

Output:-



- **DAY – 5**

-
- 1) Strings
 - 2) Array
 - 3) Array Methods
-

➔Strings

Strings can be enclosed within either single quotes, double quotes or backticks:

Example:-

```
let single = 'single-quoted';  
let double = "double-quoted";  
let backticks = `backticks`;
```

➔String Length

```
Document.write ( `My\n`.length ); // 3)
```

➔Accessing Character

We can access characters of string by using the index number.

Example:-

```
let str = `Hello`;  
// the first character  
Document.write( str[0] ); // H  
Document.write( str.at(0) ); // H
```

➔Methods

```
Document.write( 'Interface'.toUpperCase() ); // INTERFACE  
Document.write( 'Interface'.toLowerCase() ); // interface  
let str = 'Widget with id';  
document.write( str.indexOf('Widget') ); // 0  
document.write( "Widget with id".includes("Widget") ); // true  
document.write( "Widget".startsWith("Wid") ); // true, "Widget" starts with "Wid"  
document.write( "Widget".endsWith("get") ); // true, "Widget" ends with "get"  
let str = "stringify";
```

```
document.write( str.slice(0, 5) );//strin  
document.write( str.substring(2, 6) ); // "ring"
```

➔Arrays

Objects allow you to store keyed collections of values. That's fine.

But quite often we find that we need an ordered collection, where we have a 1st, a 2nd, a 3rd element and so on. For example, we need that to store a list of something: users, goods, HTML elements etc.

It is not convenient to use an object here, because it provides no methods to manage the order of elements. We can't insert a new property "between" the existing ones. Objects are just not meant for such use.

There exists a special data structure named **Array**, to store ordered collections.

➔Declaration:-

```
let arr = new Array();  
let arr = [];
```

➔Accessing:-

Can access the array element using the index numbers.

```
let fruits = ["Apple", "Orange", "Plum"];  
document.write( fruits[0] ); // Apple  
document.write ( fruits[1] ); // Orange  
document.write ( fruits[2] ); // Plum
```

➔Methods:-

It has various methods like push pop shift and unshift.

- ➔push adds an element to the end.
- ➔pop takes an element from the end.
- ➔shift takes an element from first.
- ➔unshift adds an element to the first.

Example:-

```
let fruits = ["Apple", "Orange", "Plum"];  
fruits.pop();  
document.write( fruits );  
fruits.push("Pear");  
document.write( fruits );  
fruits.shift();  
document.write( fruits );  
fruits.unshift('Apple');  
document.write( fruits );  
document.write(fruits.length);
```

Output:-

```
Apple,Orange  
Apple,Orange,Pear  
Orange,Pear  
Apple,Orange,Pear  
3
```

- **DAY – 6**

1) Date and Time

2) Json

➔Date and Time

Let's meet a new built-in object: Date. It stores the date, time and provides methods for date/time management.

For instance, we can use it to store creation/modification times, to measure time, or just to print out the current date.

➔Creation:-

To create a new Date object call new Date() with one of the following arguments:

new Date();

Example:-

```
Let date = new Date();
```

```
Document.write(date);
```

Wed Aug 16 2023 12:50:07 GMT+0530 (India Standard Time)

➔Access date components.

➔getFullYear()

Get the year (4 digits)

➔getMonth()

Get the month, from 0 to 11.

➔getDate()

Get the day of month, from 1 to 31, the name of the method does look a little bit strange.

➔getHours(), getMinutes(), getSeconds(), getMilliseconds()

Get the corresponding time components.

➔Example:-

```
let date = new Date();
```

```
document.write(date.getFullYear());  
document.write("<br>");  
document.write(date.getMonth());  
document.write("<br>");  
document.write(date.getDate());
```

→Output:-

2023
7
16

→Setting Date Components.

The following methods allow to set date/time components:

```
setFullYear(year, [month], [date])  
setMonth(month, [date])  
setDate(date)  
setHours(hour, [min], [sec], [ms])  
setMinutes(min, [sec], [ms])  
setSeconds(sec, [ms])
```

→Example:-

```
let today = new Date();  
today.setHours(0);  
document.write(today);
```

→Output:-

Wed Aug 16 2023 00:59:07 GMT+0530 (India Standard Time)

➔Json

JSON(Javascript Object Notation) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.

➔Json Methods.

JSON.stringify to convert objects into JSON.

JSON.parse to convert JSON back into an object.

➔Example:-

```
let student = {  
  name: "Kevish",  
  age: 21,  
  isAdmin: false,  
  courses: ["html", "css", "js"],  
};  
  
let json = JSON.stringify(student);  
document.write(json);
```

Output:-

```
{"name":"Kevish","age":21,"isAdmin":false,"courses":["html","css","js"]}
```

- **DAY – 7**

-
- 1) Recursion
 - 2) Call, Apply, Bind
 - 3) Arrow Function.
-

➔ Recursion

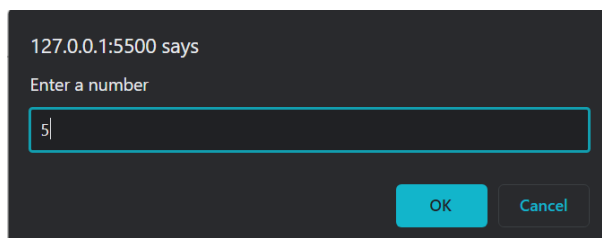
Recursion is a programming pattern that is useful in situations when a task can be naturally split into several tasks of the same kind, but simpler. Or when a task can be simplified into an easy action plus a simpler variant of the same task. Or, as we'll see soon, to deal with certain data structures.

When a function solves a task, in the process it can call many other functions. A partial case of this is when a function calls itself. That's called recursion.

➔ Example:- (Factorial of n)

```
let n = +prompt("Enter a number");  
function factorial(n){  
  if(n==1){  
    return 1  
  }else{  
    return n * factorial(n-1)  
  }  
}  
factorial(n)  
document.write(factorial(n))
```

➔ Output:-



120

➔Call, Bind and Apply

➔Call is a function that helps you change the context of the invoking function. In layperson's terms, it helps you replace the value of this inside a function with whatever value you want.

➔Apply is very similar to the call function. The only difference is that in apply you can pass an array as an argument list.

➔Bind is a function that helps you create another function that you can execute later with the new context of this that is provided.

➔Example:-

```
const kevis = {  
  name: "Kevish",  
  age: 21,  
};  
  
function greeting() {  
  document.write(`Hi, I am ${this.name} and I am ${this.age} years old`);  
}  
  
const greetingMe = greeting.bind(kevis);  
greetingMe();
```

➔Output:-

Hi, I am Kevish and I am 21 years old

➔Arrow Function

There's another very simple and concise syntax for creating functions, that's often better than Function Expressions.

It's called "arrow functions", because it looks like this:

```
let func = (arg1, arg2, ..., argN) => expression;
```

This creates a function func that accepts arguments arg1..argN, then evaluates the expression on the right side with their use and returns its result.

In other words, it's the shorter version of:

```
let func = function(arg1, arg2, ..., argN) {  
  return expression;  
};
```

➔Example:-

```
let sum = (a, b) => a + b;  
document.write( sum(1, 2) ); // 3
```

➔Output:-

3

- **DAY – 8**

1) Error Handling

➔Try...Catch

No matter how great we are at programming, sometimes our scripts have errors. They may occur because of our mistakes, an unexpected user input, an erroneous server response, and for a thousand other reasons.

Usually, a script “dies” (immediately stops) in case of an error, printing it to console.

But there’s a syntax construct try...catch that allows us to “catch” errors so the script can, instead of dying, do something more reasonable.

➔The “try...catch” syntax

The try...catch construct has two main blocks: try, and then catch:

```
try {  
  // code...  
} catch (err) {  
  // error handling  
}
```

It works like this:

First, the code in try { ... } is executed.

If there were no errors, then catch (err) is ignored: the execution reaches the end of try and goes on, skipping catch.

If an error occurs, then the try execution is stopped, and control flows to the beginning of catch (err). The err variable (we can use any name for it) will contain an error object with details about what happened.

➔Example:-

```
let json = "{ bad json }";  
  
try {  
  let user = JSON.parse(json);  
  alert( user.name );  
} catch (err) {
```

```
Document.write( "Our apologies, the data has errors, we'll try to request it one more  
time." );  
  
Document.write( err.name );  
  
Document.write( err.message );  
  
}
```

→Output

SyntaxError
Expected property name or '}' in JSON at position 2

➔Try catch Finally

The try...catch construct may have one more code clause: finally.

If it exists, it runs in all cases:

after try, if there were no errors,

after catch, if there were errors.

The extended syntax looks like this:

```
try {  
    ... try to execute the code ...  
} catch (err) {  
    ... handle errors ...  
} finally {  
    ... execute always ...  
}
```

→Example:-

```
try {  
    document.write( 'try' );  
    if (confirm('Make an error?')) BAD_CODE();  
} catch (err) {
```

```
        document.write ( 'catch' );  
    } finally {  
        document.write ( 'finally' );  
    }
```

→Output:-

```
try  
catch  
finally
```

- **DAY – 9**

-
- 1) Promise
 - 2) Async Await
-

➔Promise

- 1) A “producing code” that does something and takes time. For instance, some code that loads the data over a network. That’s a “singer”.
- 2) A “consuming code” that wants the result of the “producing code” once it’s ready. Many functions may need that result. These are the “fans”.
- 3) A promise is a special JavaScript object that links the “producing code” and the “consuming code” together. In terms of our analogy: this is the “subscription list”. The “producing code” takes whatever time it needs to produce the promised result, and the “promise” makes that result available to all of the subscribed code when it’s ready.

➔The constructor syntax for a promise object is:

```
let promise = new Promise(function(resolve, reject) {  
    // executor  
});
```

➔ The promise object returned by the new Promise constructor has these internal properties:

- ➔ state — initially "pending", then changes to either "fulfilled" when resolve is called or "rejected" when reject is called.
- ➔ result — initially undefined, then changes to value when resolve(value) is called or error when reject(error) is called.

➔Example:-

```
let promise = new Promise(resolve => {  
    setTimeout(() => resolve("done!"), 1000);  
});
```

```
promise.then(alert); // shows "done!" after 1 second
```

➔Async/Await

There's a special syntax to work with promises in a more comfortable fashion, called "async/await". It's surprisingly easy to understand and use.

→ Async:-

Let's start with the async keyword. It can be placed before a function, like this:

```
async function f() {  
  return 1;  
}
```

The word "async" before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically.

→ Await:-

The syntax:

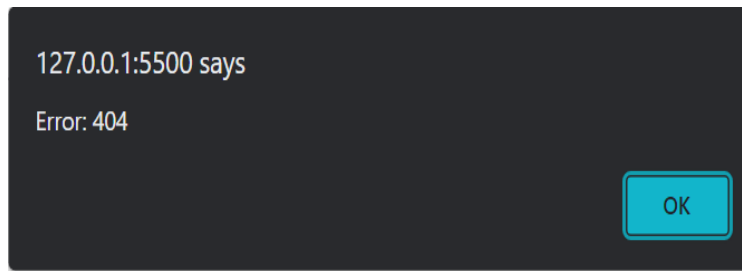
```
let value = await promise;
```

The keyword await makes JavaScript wait until that promise settles and returns its result.

→ Example:-

```
async function loadJson(url) {  
  let response = await fetch(url);  
  if (response.status === 200) {  
    let json = await response.json();  
    return json;  
  }  
  throw new Error(response.status);  
}  
  
loadJson('https://randomapi/no-such-user.json')  
  .catch(alert);
```

→ Output:-



- **DAY – 10**

1) JS Task

2) Introduction to React JS

➔Js Task:-

➔Question:-

Employee
id, name, age, salary

Eg:

```
[{  
  id: 101,  
  name: Abc,  
  age: 48,  
  salary: 50000  
}, {  
  id: 202,  
  name: Xyz,  
  age: 24,  
  salary: 40000  
}]
```

1. Find employees with name starting with `S`
2. Find employees with age > 50
3. Find employee with id 101
4. Increase salary of all employees with 10,000
5. Add a new employee at position 3
6. Delete employee at position 2
7. Find index of employee with id 202
8. Find if some employee has salary less than 100
9. Create a new array and add an additional property upperName which will have all names in uppercase

➔Solution:-

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <title>Array Method Tasks</title>  
  </head>  
  <body>  
    <h1>Array Methods Tasks</h1>  
    <script>
```

```
let employee = [  
  {  
    id: 101,  
    name: "Akhil",  
    age: 25,  
    salary: 80000,  
  },  
  {  
    id: 102,  
    name: "kevish",  
    age: 25,  
    salary: 75000,  
  },  
  {  
    id: 103,  
    name: "nikhil",  
    age: 55,  
    salary: 55000,  
  },  
  {  
    id: 202,  
    name: "Sarthak",  
    age: 25,  
    salary: 100000,  
  },  
  {  
    id: 105,  
    name: "vatsal",  
    age: 25,  
    salary: 90000,  
  },  
  {  
    id: 106,  
    name: "Suresh",  
    age: 80,  
    salary: 90,  
  },  
  {  
    id: 107,  
    name: "Shami",  
    age: 25,  
    salary: 45000,  
  },  
  {  
    id: 108,  
    name: "Srivalli",  
    age: 25,  
    salary: 500000,  
  },  
];
```

```
/* *task 1 find employee, name starting with S */
document.write("<h4>Task 1 : find employee, name starting with
S</h4>");
let empNameWithS = employee.filter(
  (value) => value.name.charAt(0) == "S"
);
empNameWithS.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});

/* *task 2 find employee, age is greater than 50 */
document.write(
  "<h4>Task 2 : find employee, age is greater than 50 </h4>"
);

let empGtAge = employee.filter((value) => value.age > 50);
empGtAge.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});

/* *task 3 find employee, id = 101 */
document.write("<h4>Task 3 : find employee, id = 101 </h4>");

let empID = employee.filter((value) => value.id === 101);
empID.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});

/* *task 4 increase salary with 10000 */
document.write("<h4>Task 4 : increase salary with 10000 </h4>");

employee.map((value) => {
  value.salary += 10000;
});

employee.map((value) => {
  document.write("Id: " + value.id + " ");
```

```
document.write("Name: " + value.name + " ");
document.write("Age: " + value.age + " ");
document.write("Salary: " + value.salary + " ");
document.write("<br>");
});

/* *task 5 add employee at position 3 */
document.write("<h4>Task 5 : add employee at position 3 </h4>");

let newEmployee = {
  id: 150,
  name: "Bahubali",
  age: 36,
  salary: 120000,
};
employee.splice(2, 0, newEmployee);
employee.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});

/* *task 6 delete employee at position 2 */
document.write("<h4>Task 6 : delete employee at position 2 </h4>");

employee.splice(1, 1);
employee.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});

/* *task 7 find index of employee with id 202 */
document.write("<h4>Task 7 : find index of employee with id 202
</h4>");

let indOf = employee.findIndex((value) => {
  return value.id === 202;
});
if (indOf === -1) {
  document.write("No employee has given id");
} else {
  document.write("Index is " + indOf);
}

/* *task 8 find an employee having salary less than 100 */
```

```
document.write(
  "<h4>Task 8 : find an employee having salary less than 100 </h4>"
);

let lessThanSalary = employee.filter((value) => value.salary < 100);
if (lessThanSalary.length > 0) {
  document.write(
    "Number of employee having salary less than 100 are :- " +
    lessThanSalary.length
  );
} else {
  document.write("Everyone has salary greater than 100");
}

/* * task 9 Create a new array and add an additional property upperName
which will have all names in uppercase */
document.write(
  "<h4>Task 9 : Create a new array and add an additional property
upperName which will have all names in uppercase </h4>"
);

let changeArray = employee.map((value, index) => {
  return { uprName: value.name.toUpperCase(), ...value };
});
changeArray.map((value) => {
  document.write("Id: " + value.id + " ");
  document.write("uprName: " + value.uprName + " ");
  document.write("Name: " + value.name + " ");
  document.write("Age: " + value.age + " ");
  document.write("Salary: " + value.salary + " ");
  document.write("<br>");
});
// console.log(changeArray)
</script>
</body>
</html>
```

→Output:-

Array Methods Tasks

Task 1 : find employee, name starting with S

Id: 202 Name: Sarthak Age: 25 Salary: 100000

Id: 106 Name: Suresh Age: 80 Salary: 90

Id: 107 Name: Shami Age: 25 Salary: 45000

Id: 108 Name: Srivalli Age: 25 Salary: 500000

Task 2 : find employee, age is greater than 50

Id: 103 Name: nikhil Age: 55 Salary: 55000

Id: 106 Name: Suresh Age: 80 Salary: 90

Task 3 : find employee, id = 101

Id: 101 Name: Akhil Age: 25 Salary: 80000

Task 4 : increase salary with 10000

Id: 101 Name: Akhil Age: 25 Salary: 90000

Id: 102 Name: kevish Age: 25 Salary: 85000

Id: 103 Name: nikhil Age: 55 Salary: 65000

Id: 202 Name: Sarthak Age: 25 Salary: 110000

Id: 105 Name: vatsal Age: 25 Salary: 100000

Id: 106 Name: Suresh Age: 80 Salary: 10090

Id: 107 Name: Shami Age: 25 Salary: 55000

Id: 108 Name: Srivalli Age: 25 Salary: 510000

Task 6 : delete employee at position 2

Id: 101 Name: Akhil Age: 25 Salary: 90000
Id: 150 Name: Bahubali Age: 36 Salary: 120000
Id: 103 Name: nikhil Age: 55 Salary: 65000
Id: 202 Name: Sarthak Age: 25 Salary: 110000
Id: 105 Name: vatsal Age: 25 Salary: 100000
Id: 106 Name: Suresh Age: 80 Salary: 10090
Id: 107 Name: Shami Age: 25 Salary: 55000
Id: 108 Name: Srivalli Age: 25 Salary: 510000

Task 7 : find index of employee with id 202

Index is 3

Task 8 : find an employee having salary less than 100

Everyone has salary greater than 100

Task 9 : Create a new array and add an additional property upperName which will have all names in uppercase

Id: 101 uprName: AKHIL Name: Akhil Age: 25 Salary: 90000
Id: 150 uprName: BAHUBALI Name: Bahubali Age: 36 Salary: 120000
Id: 103 uprName: NIKHIL Name: nikhil Age: 55 Salary: 65000
Id: 202 uprName: SARTHAK Name: Sarthak Age: 25 Salary: 110000
Id: 105 uprName: VATSAL Name: vatsal Age: 25 Salary: 100000
Id: 106 uprName: SURESH Name: Suresh Age: 80 Salary: 10090
Id: 107 uprName: SHAMI Name: Shami Age: 25 Salary: 55000
Id: 108 uprName: SRIVALLI Name: Srivalli Age: 25 Salary: 510000

➔Introduction to React JS**➔What is React JS?**

The React.js framework is an open-source JavaScript framework and library developed by Facebook. It's used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript.

In React, you develop your applications by creating reusable components that you can think of as independent Lego blocks. These components are individual pieces of a final interface, which, when assembled, form the application's entire user interface.

React's primary role in an application is to handle the view layer of that application just like the V in a model-view-controller (MVC) pattern by providing the best and most efficient rendering execution. Rather than dealing with the whole user interface as a single unit, React.js encourages developers to separate these complex UIs into individual reusable components that form the building blocks of the whole UI. In doing so, the ReactJS framework combines the speed and efficiency of JavaScript with a more efficient method of manipulating the DOM to render web pages faster and create highly dynamic and responsive web applications.

➔Requirements for React JS**➔ Node JS**

- ➔ VS code
- ➔ Basic Knowledge of JS
- ➔ HTML
- ➔ CSS

- **DAY – 11**

-
- 1) First React Program
 - 2) What is JSX?
 - 3) Props
-

➔ Create React Project:-

To create a project called my-app, run this command:

npx create-react-app my-app

➔ To run write **npm start** in the terminal

➔ App.js

```
import './App.css';
function App() {
  return (
    <div className="App">

      <h1>HelloWorld</h1>
    </div>
  );
}
export default App;
```

➔ Output:-

HelloWorld

➔ What is JSX?

Consider this variable declaration:

```
const element = <h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

JSX produces React “elements”. We will explore rendering them to the DOM in the next section. Below, you can find the basics of JSX necessary to get you started.

➔Example:-

```
import './App.css';
function App() {
  return (
    <div className="App">

      <h1>HelloWorld</h1>
    </div>
  );
}
export default App;
```

➔Output:-

HelloWorld

➔Props

React Props are like function arguments in JavaScript and attributes in HTML.

To send props into a component, use the same syntax as HTML attributes:

➔Example:-

➔App.js

```
import './App.css';

import Product from './components/Product';

function App() {
  const product = [
    {
      title: "Product 1",
      price: "10",
      description: "First product",
    },
    {
      title: "Product 2",
      price: "20",
      description: "Second product",
    },
  ];
  return (
    <div className="App">

      <h1>My Demo Shop</h1>
      <Product
        title={product[0].title}
        price={product[0].price}
        description={product[0].description}
      ></Product>
      <Product
        title={product[1].title}
        price={product[1].price}
        description={product[1].description}
      ></Product>

    </div>
  );
}

export default App;
```

➔Product.js

```
import React from "react";
import "../product.css"
export default function Product(props) {
  return (
    <article className="product">
      <h2>{props.title}</h2>
      <p className="price">${props.price}</p>
      <p>{props.description}</p>
    </article>
  );
}
```

→product.css

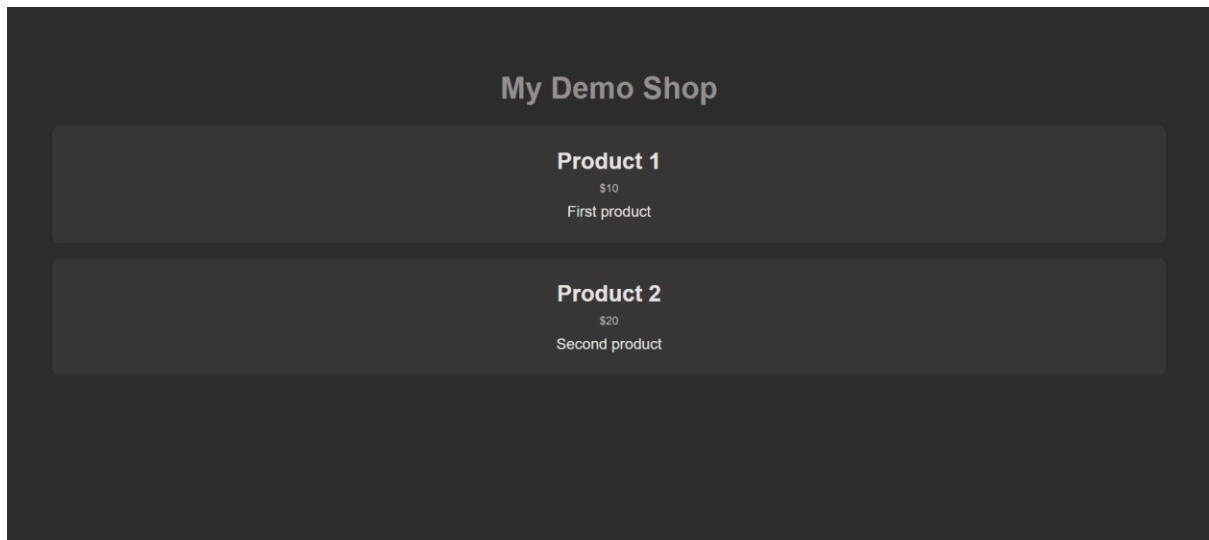
```
body {
  font-family: sans-serif;
  margin: 0;
  padding: 3rem;
  background-color: #2d2c2c;
  color: #959090;
}
```

```
.product {
  margin: 1rem 0;
  padding: 1rem;
  background-color: #373535;
  color: #e7e4e4;
  border-radius: 8px;
}
```

```
.product h2,
.product p {
  margin: 0.5rem 0;
}
```

```
.price {
  font-size: 0.75rem;
  color: #bab6b6;
}
```

→Output:-



- **DAY – 12**

1) Hooks

➔ Hooks

➔ useState:-

```
import React, { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

➔ useEffect:-

```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

- **DAY – 13**

1) Expense Project

➔Expense Project:-

Here everyone can enter daily expense and can filter according to year wise. By this project concept like useState and useEffect become more stronger and concept of props was really usefull in this.

➔App.js

```
import logo from "../logo.svg";
import "../App.css";
import Header from "../Components/Header";
import DisplayExpense from "../Components/Display/DisplayExpense";
import { useState, useEffect } from "react";
import Filter from "../Components/Display/Filter";

function App() {
  const [arr, setarr] = useState([
    { title: "vadapau", amount: "20", date: "8/1/2023, 5:30:00 AM" },
  ]);
  const [filterArr, setFilterArr] = useState([]);
  const [year, setYear] = useState("");
  const [yearArr, setYearArr] = useState();
  useEffect(() => {
    setYearArr(
      arr?.map((value) => {
        return new Date(value.date).getFullYear().toString();
      })
    );
  }, [arr]);

  const newHandler = (obj) => {
    setarr((prev) => {
      return [obj, ...prev];
    });

    let demoyear = new Date(obj.date).getFullYear().toString();
    // console.log(year)
    if (demoyear === year) {
      setFilterArr((prev) => {
        return [obj, ...prev];
      });
    }
  };

  const filterHandler = (array, selectedYear) => {
    // console.log(array)
```

```

    setFilterArr(array);
    setYear(selectedYear);
  };

  return (
    <>
      <Header getNew={newHandler} />
      <Filter
        expense={arr}
        filter={filterHandler}
        getYearArr={new Set(yearArr)}
      />
      {year == "select year" ? (
        <DisplayExpense expense={arr} />
      ) : (
        <DisplayExpense expense={filterArr} />
      )}
    </>
  );
}

export default App;

```

→Form.js

```

import React, { useState } from 'react'
import './Form.css'
const Form = (props) => {
  const [title, setTitle] = useState("")
  const [amount, setAmount] = useState("")
  const [tarikh, setTarikh] = useState("")
  const titleHandler = (e) =>{
    setTitle(e.target.value)
  }
  const amountHandler = (e) =>{
    setAmount(e.target.value)
  }
  const tarikhHandler = (e) =>{
    setTarikh(e.target.value)
  }
  const submitHandler = (e) =>{
    e.preventDefault()
    let newObj = {
      title:title,
      amount:amount,
      date:new Date(tarikh).toLocaleString()
    }
    // console.log(newObj)
    props.getNewData(newObj)
  }
}

```



```

        setAmount("")
        setTarikh("")
        setTitle("")
    }
    return (
    <div>
    <form className="formDiv" onSubmit={submitHandler}>
    <label className="formLabel">Enter Title</label>
    <input
    type="text"
    onChange={titleHandler}
    value={title}
    placeholder="Enter Title"
    />
    <label className="formLabel">Enter Amount</label>
    <input
    type="text"
    onChange={amountHandler}
    value={amount}
    placeholder="Enter Description"
    />
    <label className="formLabel">Pick a Date</label>
    <input type="date" value={tarikh} onChange={tarikhHandler} />
    <button className="formBtn">Submiit</button>
    </form>
    </div>
    );
}

```

export default Form

→Header.js

```

import React, { useState } from "react";
import Form from "../Form";

const Header = (props) => {
    const [showForm, setShowForm] = useState(false);
    const newDataHandler = (data) =>{
        console.log(data)
        props.getNew(data)
    }
    return (
    <div style={{ textAlign: "center" }}>
    <h2 style={{ color: "#DDE6ED" }}>
    Enter Data/Hide Form
    <button
    className="showBtn"
    onClick={() => {
        setShowForm(!showForm);
    }}
    />
    </h2>
    </div>
    );
}

```

```

    }}
  >
    {showForm? "\u2191": "\u2193"}
  </button>
</h2>
{showForm ? <Form getNewData={newDataHandler}/> : null}
</div>
);
};

```

export default Header;

→ Filter.js

```

import React, { useEffect, useState } from "react";
import "../DisplayExpense.css";
const Filter = ({ expense, filter, getYearArr }) => {
  const [selectedYear, setSelectedYear] = useState('select year');
  let filteredArray = []
  const chageYearHandler = (e) => {
    setSelectedYear(e.target.value)
  }
  useEffect(() => {
    if(expense.length > 0){
      filteredArray = expense.filter((value) => {
        let year = new Date(value.date).getFullYear().toString();
        // console.log(year)
        // console.log(selectedYear)
        return selectedYear === year;
      });
      // console.log(filteredArray)
      filter(filteredArray, selectedYear);
    }
  }, [selectedYear])

  return (
    <div className="dispFilter">
      <h3>Select Year</h3>
      <select
        className="selectDisp"
        onChange={chageYearHandler}
        value={selectedYear}
      >
        <option value="select year">Select year</option>

```

```

      <option value="2020">2020</option>
      <option value="2021">2021</option>
      <option value="2022">2022</option>
      <option value="2023">2023</option>

    </select>
  </div>
);
};

```

export default Filter;

→DisplayExpenseItem.js

```

import React from 'react'
import './DisplayExpense.css'
const DisplayExpenseItem = ({title,amount,date}) => {
  const getYear = new Date(date).getFullYear().toString()
  const getMonth = new Date(date).getMonth().toString()
  const getDate = new Date(date).getDate().toString()
  return (
    <div className="itemDisp">
      <div className="dateDisp">
        {getDate + " / " + (parseInt(getMonth) + 1) + " / " + getYear}
      </div>
      <div className="titleDisp">{title.toUpperCase()}</div>
      <div className="dateDisp">₹{amount}</div>
    </div>
  );
}

```

export default DisplayExpenseItem

→DisplayExpense.js

```

import React from "react";
import "./DisplayExpense.css";
import DisplayExpenseItem from "./DisplayExpenseItem";
const DisplayExpense = ({expense}) => {
  return (
    <div className="dispDiv">
      {expense.length>0 ? (
        expense.map((val, index) => (
          <DisplayExpenseItem
            key={index}
            title={val.title}
            amount={val.amount}

```

```
        date={val.date}
      />
    ))
  ): (
    <h2 style={{ color: "#116A7B" ,textAlign:"center"}}>No Items Yet...</h2>
  )}
  { /* {console.log(expense)} */ }
</div>
);
};
```

export default DisplayExpense;

→Form.css

```
.formDiv {
  height: auto;
  width: 40%;
  border-radius: 20px;
  background-color: #526d82;
  margin-left: auto;
  margin-right: auto;
  justify-content: center;
  align-items: center;
  display: flex;
  flex-direction: column;
  padding: 15px;
}
```

```
.formLabel {
  color: #dde6ed;
  font-size: 20px;
  margin-top: 10px;
}
```

```
.formDiv input {
  background-color: #9db2bf;
  color: #27374d;
  border: 1px solid #dde6ed;
  border-radius: 10px;
  font-size: 20px;
  padding: 10px;
}
.formDiv input:focus {
  outline: none;
}
```

```
.formBtn {
  margin-top: 10px;
  font-size: 20px;
```

```
background-color: #27374d;
color: #dde6ed;
border: 1px solid #9db2bf;
border-radius: 10px;
padding: 10px;
margin-bottom: 15px;
}
```

```
.showBtn {
background-color: #dde6ed;
color: #27374d;
border: 1px solid #9db2bf;
border-radius: 10px;
font-size: 30px;
margin-left: 10px;
padding: 8px;
}
```

→DisplayExpense.css

```
.dispDiv {
height: auto;
width: 75%;
border-radius: 20px;
background-color: #CDC2AE;
margin-left: auto;
margin-right: auto;
align-items: center;
display: flex;
flex-direction: column;
padding: 15px;
margin-top: 10px;
margin-bottom: 15px;
}
```

```
.itemDisp{
height: 15vh;
width: 90%;
border-radius: 10px;
background-color:#116A7B;
margin-left: auto;
margin-right: auto;
align-items: center;
display: flex;
flex-direction: row;
align-items: center;
row-gap: 10%;
padding: 15px;
margin-top: 10px;
margin-bottom: 15px;
}
```

```
}

.dateDisp{
  border:2px solid #ECE5C7;
  padding: 20px;
  font-size: 20px;
  box-shadow: #116A7B;
  color: #ECE5C7 ;
  width: 20%;
  text-align: center;
}

.titleDisp{
  padding: 20px;
  font-size: 20px;
  color: #ECE5C7 ;
  width: 70%;
  text-align: center;
}

.dispFilter{
  height: auto;
  width: 75%;
  border-radius: 5px;
  background-color: #C2DEDC;
  margin-left: auto;
  margin-right: auto;
  align-items: center;
  display: flex;
  flex-direction: row;
  justify-content: center;
  padding: 15px;
  margin-top: 10px;
  margin-bottom: 15px;
  gap: 50%;
}

.selectDisp{
  background-color: #374f5e;
  color: #d5e0f1;
  border: 1px solid #dde6ed;
  border-radius: 10px;
  font-size: 15px;
  padding: 15px;
}

.selectDisp:focus{
  outline: none;
}
```

→Output:-


The image displays two screenshots of a web application interface, demonstrating a form that can be toggled between a collapsed and an expanded state.

Top Screenshot (Collapsed State):

- Header: "Enter Data/Hide Form" with a downward arrow icon.
- Form Container (Light Blue):
 - Label: "Select Year"
 - Dropdown: "Select year" with a downward arrow.
- Data Row (Teal):
 - Input: "1 / 8 / 2023"
 - Text: "VADAPAU"
 - Input: "₹20"

Bottom Screenshot (Expanded State):

- Header: "Enter Data/Hide Form" with an upward arrow icon.
- Form Container (Light Blue):
 - Label: "Select Year"
 - Dropdown: "Select year" with a downward arrow.
- Expanded Form (Dark Blue):
 - Label: "Enter Title"
 - Input: "Enter Title"
 - Label: "Enter Amount"
 - Input: "Enter Description"
 - Label: "Pick a Date"
 - Input: "dd-mm-yyyy" with a calendar icon
 - Button: "Submit"

Enter Data/Hide Form 

Select Year

Select year ▾

1 / 8 / 2023


DABELI

₹500

1 / 8 / 2023

VADAPAU

₹20

Enter Data/Hide Form 

Select Year

2021 ▾

No Items Yet...

- **DAY – 14**

1) Styled components

2) useContext

➔Styled Components

styled-components is the result of wondering how we could enhance CSS for styling React component systems. By focusing on a single use case we managed to optimize the experience for developers as well as the output for end users.

Apart from the improved experience for developers, styled-components provides:

Automatic critical CSS: styled-components keeps track of which components are rendered on a page and injects their styles and nothing else, fully automatically. Combined with code splitting, this means your users load the least amount of code necessary.

No class name bugs: styled-components generates unique class names for your styles. You never have to worry about duplication, overlap or misspellings.

Easier deletion of CSS: it can be hard to know whether a class name is used somewhere in your codebase. styled-components makes it obvious, as every bit of styling is tied to a specific component. If the component is unused (which tooling can detect) and gets deleted, all its styles get deleted with it.

Simple dynamic styling: adapting the styling of a component based on its props or a global theme is simple and intuitive without having to manually manage dozens of classes.

Painless maintenance: you never have to hunt across different files to find the styling affecting your component, so maintenance is a piece of cake no matter how big your codebase is.

Automatic vendor prefixing: write your CSS to the current standard and let styled-components handle the rest.

You get all of these benefits while still writing the CSS you know and love, just bound to individual components.

➔Installation

Installing styled-components only takes a single command and you're ready to roll:

npm install styled-components

➔useContext Hook

React context provides data to components no matter how deep they are in the components tree. The context is used to manage global data, e.g. global state, theme, services, user settings, and more.

➔ A. Creating the context

The built-in factory function `createContext(default)` creates a context instance:

```
// context.js
```

```
import { createContext } from 'react';
```

```
export const Context = createContext('Default Value');
```

The factory function accepts one optional argument: the default value.

➔ B. Providing the context

`Context.Provider` component available on the context instance is used to provide the context to its child components, no matter how deep they are.

To set the value of context use the `value` prop available on the

```
<Context.Provider value={value} />:
```

```
import { Context } from './context';
```

```
function Main() {
```

```
  const value = 'My Context Value';
```

```
  return (
```

```
    <Context.Provider value={value}>
```

```
      <MyComponent />
```

```
    </Context.Provider>
```

```
  );
```

```
}
```

Again, what's important here is that all the components that'd like later to consume the context have to be wrapped inside the provider component.

If you want to change the context value, simply update the `value` prop.

➔ C. Consuming the context

Consuming the context can be performed in 2 ways.

The first way, the one I recommend, is to use the useContext(Context) React hook:

```
import { useContext } from 'react';  
import { Context } from './context';  
function MyComponent() {  
  const value = useContext(Context)  
  return <span>{ value}</span>;  
}
```

- **DAY – 15**

1) Login Project

➔Login Project

➔This project helps me to understand useContext hook and styled components.

➔Styledcomponents.js

```
import react from 'react'
import styles from 'styled-components'

export const Div = styles.div`
  height: 100vh;
  width:100%;
  background-color:#002333;
  display:flex;
  align-items:center;
  justify-content:center;
`;

export const Card = styles.div`
  width:50%;
  height:55%;
  background-color:#FFFFFF;
  box-shadow:0px 0px 25px 4px #DEEFE7;
  border-radius:10px;
  display:flex;
  flex-direction:column;
  align-items:center;
  justify-content:center;
  gap:5%;
  @media (max-width: 768px) {
```

```
        width:80%;
        height:65%
    };
    @media (max-width:425px){
        width:80%;
        height:65%;
    }
`;

export const Input = styles.input`
    padding:5px;
    border-radius:5px;
    border: 2px solid #0f804b;
    width:70%;
    height:10%;
    font-size:15px;
    &:focus{
        outline:none;
        box-shadow:0px 0px 10px 4px #86d1ae;
    }
`;

export const Button = styles.button`
    border:3px solid #0f804b;
    background-color:#FFFFFF;
    padding:15px 30px 15px 30px;
    font-size:15px;
    transition:all 0.5s;
    border-radius:5px;
    margin-bottom:20px;
    margin-top:10px;
    &:hover{
```

```
border:3px solid #0f804b;
background-color:#0f804b;
color:white;
}
@media(max-width:786px){
width:70%;
border:3px solid #0f804b;
background-color:#0f804b;
color:white;
}
`;

export const Typography = styles.text`
color:red;
font-weight:bold;
`

export const Label = styles.label`
letter-spacing:px;
color:#002333;
margin-top:12px;
font-size:25px;
font-weight:bold; `;
```

→App.js

```
import { useContext, useEffect, useRef, useState } from "react";
import {
  Button,
  Card,
  Div,
  Input,
  Label,
  Typography,
```

```
    } from "../components/StyledComponent";
    import context from "../components/Context";

    function App() {
      useEffect(() => {
        if (localStorage.getItem("LoggedIn")) {
          onLoginHandler("", "", true)
        }
      }, []);

      const { isLoggedIn, isErrorInInput, onLoginHandler, onLogoutHandler } =
        useContext(context);
      const emailRef = useRef();
      const passRef = useRef();
      const logandarHandler = () => {
        let uemail = emailRef.current.value;
        let upass = passRef.current.value;
        onLoginHandler(uemail, upass);
      };
      const logbaharHandler = () => {
        // console.log("logbahar")
        onLogoutHandler();
      };

      return (
        <
          <!isLoggedIn ? (
            <Div>
              <Card>
                <Label>Email</Label>

```

```

    <Input placeholder="Email" ref={emailRef} type="email" />
    <Label>Password</Label>
    <Input placeholder="Password" ref={passRef} type="password" />
    {isErrorInInput ? (
      <Typography>Invalid Credentials</Typography>
    ) : null}
    <Button onClick={logandarHandler}>Login</Button>
  </Card>
</Div>
): (
  <Div>
    <Card>
      <Button onClick={logbaharHandler}>LogOut</Button>
    </Card>
  </Div>
)}
</>
);
}
export default App;

```

→Context.js

```

import { createContext,useEffect, useState } from "react";
const context = createContext({
  isLoggedIn:false,
  isErrorInInput:false,
  onLoginHandler:()=>{ },
  onLogoutHandler:()=>{ },
  // onErrorHandler:()=>{ },
});

```



```
export const ContextProvider = ({children}) =>{

  const [errorInInput,setErrorInInput] = useState(false);
  const [loggedIn,setLoggedIn] = useState(false);
  const loginHandler = (email,password,fromUse)=>{

    if(email.includes('@')&&email.includes('.com')&&email.length>0&&password.length>=6){
      setLoggedIn(true)
      setErrorInInput(false)
      localStorage.setItem("LoggedIn",true)
    }else if(fromUse){
      setLoggedIn(true)
    }
    else{
      setErrorInInput(true)
    }
  }

  const logoutHandler = ()=>{
    console.log("console")
    setLoggedIn(false)
    localStorage.removeItem("LoggedIn")
  }

  return (
    <context.Provider
      value={{
        isErrorInInput: errorInInput,
        isLoggedIn: loggedIn,
        onLoginHandler: loginHandler,
        onLogoutHandler:logoutHandler,
```

```
    }}  
  >  
    {children}  
  </context.Provider>  
)  
}  
export default context;
```

→index.js

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import './index.css';  
import App from './App';  
import reportWebVitals from './reportWebVitals';  
import { ContextProvider } from './components/Context'  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
  <React.StrictMode>  
    <ContextProvider>  
      <App />  
    </ContextProvider>  
  </React.StrictMode>  
>);  
reportWebVitals();
```

→Output:-

The image displays two screenshots of a login form interface. The form is centered on a dark blue background and consists of a white rounded rectangle containing the following elements:

- Email**: A label above a text input field.
- Password**: A label above a text input field.
- Login**: A button located below the password field.

The top screenshot shows the form in its initial state with empty input fields. The bottom screenshot shows the form after an attempt to log in with incorrect credentials, with the text **Invalid Credentials** displayed in red below the password field.

The image shows a login interface on a dark blue background. It is divided into two horizontal sections. The top section is a white rounded rectangle containing the following elements:

- Email**: A label above a text input field containing the email address "Kevish12@gamil.com".
- Password**: A label above a password input field containing six dots ".....".
- Login**: A button located below the password field.

The bottom section is another white rounded rectangle, centered below the first one, containing a single button:

- LogOut**: A button located in the center of the bottom section.

Conclusion

Overall analysis of Summer Internship

It was a great internship, got to learn a lot like what is Industry De- fined Project (IDP), how to work in organized manner, how to present your problems and find solutioneffectively, how to make recommending web applications, using useState, useEffect, useContext, styledcomponenets .

Project Review Meeting

Learnt how to present your project and task preformed in an effective manner and also learnt how to handle events in react js and how to work with hooks , styled components , props etc..

Summary

- Introduction of JavaScript.
- Variables in javascript.
- Operators
- Data types
- Function
- Arrow Function
- Error Handling
- Promise and async/await
- Introduction to react js
- Hooks in react
- Props and styled components
- Created expense and react project