

<b>Ex. No: 1</b>	<b>STATIC WEBSITE USING HTML, CSS</b>

## AIM

Develop a visually appealing static website with an intuitive user interface with multimedia content

- Add as text, images and videos to the webpage.
- Use CSS properties to make the form appealing.

## CODE

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>My Static Website</title>
```

```
</head>
```

```
<style>
```

```
  body {
```

```
    font-family: Arial, sans-serif;
```

```
    margin: 0;
```

```
    padding: 0;
```

```
    background-color: #f4f4f4;
```

```
    color: #333;
```

```
  }header {
```

```
    background: #0073e6;
```

```
    color: white;
```

```
padding: 20px;
text-align: center;
}header h1 {
margin: 0;
}nav ul {
list-style-type: none;
padding: 0;
}nav ul li {
display: inline;
margin-right: 20px;
}nav ul li a {
color: white;
text-decoration: none;
}section {
padding: 20px;
margin: 20px;
background: #fff;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
footer {
text-align: center;
padding: 10px;
background: #333;
color: white;
}
</style>
<body>
```

```
<header>

  <h1>Welcome to My Website</h1>

  <nav>

    <ul>

      <li><a href="#home">Home</a></li><li><a
href="#about">About</a></li><li><a href="#services">Services</a></li><li><a
href="#contact">Contact</a></li>

    </ul>

  </nav>

</header>

<section id="home">

  <h2>Home</h2>

  <p>This is a simple static website created using HTML and CSS.</p>

</section>

<section id="about">

  <h2>About</h2>

  <p>We provide high-quality content and services to our clients.</p>

</section>

<section id="services">

  <h2>Services</h2>

  <ul>

    <li>Web Development</li>

    <li>SEO Optimization</li>

    <li>Graphic Design</li>

  </ul>

</section>

<section id="contact">

  <h2>Contact Us</h2>

  <p>Email: contact@example.com</p>
```

**<p>Phone: +123 456 7890</p>**

**</section>**

**<footer>**

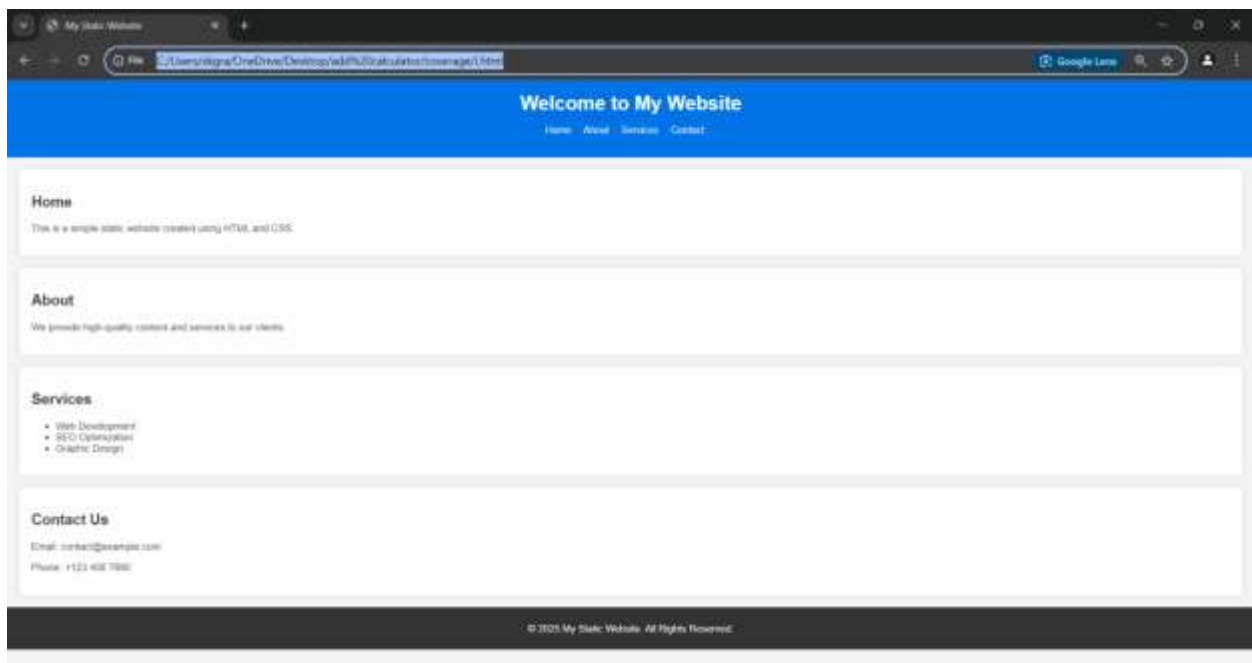
**<p>&copy; 2025 My Static Website. All Rights Reserved.</p>**

**</footer>**

**</body>**

**</html>**

## OUTPUT



## RESULT

Thus, the webpage has been created successfully.

<b>Ex. No: 2</b>	<b>FORM VALIDATION USING HTML, JAVASCRIPT</b>

**AIM:**

Create a simple form to collect the name, address, email ID and phone number from the user.

- Use CSS properties to make the form appealing.
- Write JavaScript functions for empty field validation.

**CODE**

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>User Form</title>

  <style>

    body {

      font-family: Arial, sans-serif; background: #f4f4f9; display: flex; justify-content: center; align-items: center; height: 100vh; margin: 0;

    }

    .form-container {

      background: #fff; padding: 20px; border-radius: 8px; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); width: 300px;

    }

    .form-container h2 { text-align: center; color: #333; }

    .form-group { margin-bottom: 15px; }

    .form-group label { display: block; margin-bottom: 5px; color: #555; }

    .form-group input {
```

```

width: 100%; padding: 10px; border: 1px solid #ccc; border-radius: 4px; font-size: 14px;
}

.form-group input:focus { border-color: #007bff; outline: none; box-shadow: 0 0 5px rgba(0, 123, 255,
0.5); }

.error { color: red; font-size: 12px; }

button {

width: 100%; padding: 10px; background: #007bff; color: #fff; border: none; border-radius: 4px;
cursor: pointer; font-size: 16px;

}

button:hover { background: #0056b3; }

</style>
</head>
<body>
<div class="form-container">
<h2>User Info</h2>
<form id="userForm">
<div class="form-group">
<label>Name:</label>
<input type="text" id="name">
<span id="nameError" class="error"></span>
</div>
<div class="form-group">
<label>Address:</label>
<input type="text" id="address">
<span id="addressError" class="error"></span>
</div>
<div class="form-group">

```

```

<label>Email:</label>

<input type="email" id="email">

<span id="emailError" class="error"></span>

</div>

<div class="form-group">

  <label>Phone:</label>

  <input type="text" id="phone">

  <span id="phoneError" class="error"></span>

</div>

<button type="button" onclick="validateForm()">Submit</button>

</form>

</div>

<script>

const validateForm = () => {

  let isValid = true;

  ["name", "address", "email", "phone"].forEach(id => {

    const value = document.getElementById(id).value.trim();

    const errorField = document.getElementById(id + "Error");

    errorField.textContent = value ? "" : id.charAt(0).toUpperCase() + id.slice(1) + " is required.";

    if (id === "email" && value && !/^[^\s@]+@[^\s@]+\.[^\s@]+$/ .test(value)) {

      errorField.textContent = "Invalid email."; isValid = false;

    }

    if (id === "phone" && value && !/^\\d{10}$/.test(value)) {

      errorField.textContent = "Invalid phone."; isValid = false;

    }

    if (!value) isValid = false;

```

```
});  
  
if (isValid) alert("Form submitted successfully!");  
  
};  
</script>  
</body>  
</html>
```

## OUTPUT



The screenshot displays a web form titled "User Info" centered on a light purple background. The form is a white rectangle with a thin border and a subtle drop shadow. It contains four text input fields, each preceded by a label: "Name:", "Address:", "Email:", and "Phone:". Below these fields is a blue "Submit" button. The form is currently empty, with no data entered into the fields.

## RESULT

Thus, the webpage has been created successfully



<b>Ex. No: 3</b>	<b>INTERACTIVE QUIZ USING HTML, CSS, JAVASCRIPT</b>

**AIM:**

Create an interactive quiz that allows the user to select answers to multiple-choice questions.

- Use CSS properties to make the form appealing.
- Write JavaScript functions for calculation the user's score and provide feedback based on their performance.

**CODE**

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Interactive Quiz</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      background-color: #f4f4f9;

      margin: 0;

      padding: 20px;

      display: flex;

      flex-direction: column;

      align-items: center;

    }

    .quiz-container {

      background: #ffffff;
```

```
border-radius: 10px;  
padding: 20px;  
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);  
max-width: 500px;  
width: 100%;  
}
```

```
h1 {  
  text-align: center;  
  color: #333;  
}
```

```
.question {  
  margin-bottom: 20px;  
}
```

```
.question h3 {  
  margin-bottom: 10px;  
}
```

```
.options {  
  list-style: none;  
  padding: 0;  
}
```

```
.options li {
```

```
margin-bottom: 10px;  
}
```

```
.options input {  
margin-right: 10px;  
}
```

```
button {  
background-color: #3498db;  
color: white;  
padding: 10px 20px;  
border: none;  
border-radius: 5px;  
cursor: pointer;  
font-size: 16px;  
}
```

```
button:hover {  
background-color: #2980b9;  
}
```

```
.result {  
margin-top: 20px;  
font-size: 18px;  
text-align: center;  
}
```

```
.result.success {  
    color: #2ecc71;  
}  
  
.result.failure {  
    color: #e74c3c;  
}  
  
</style>  
</head>  
<body>  
    <div class="quiz-container">  
        <h1>Interactive Quiz</h1>  
        <form id="quizForm">  
            <div class="question">  
                <h3>1. What is the capital of France?</h3>  
                <ul class="options">  
                    <li><label><input type="radio" name="q1" value="0"> Berlin</label></li>  
                    <li><label><input type="radio" name="q1" value="1"> Paris</label></li>  
                    <li><label><input type="radio" name="q1" value="0"> Madrid</label></li>  
                    <li><label><input type="radio" name="q1" value="0"> Rome</label></li>  
                </ul>  
            </div>  
            <div class="question">  
                <h3>2. What is 2 + 2?</h3>  
                <ul class="options">  
                    <li><label><input type="radio" name="q2" value="0"> 3</label></li>
```

```
<li><label><input type="radio" name="q2" value="1"> 4</label></li>

<li><label><input type="radio" name="q2" value="0"> 5</label></li>

<li><label><input type="radio" name="q2" value="0"> 6</label></li>

</ul>

</div>

<div class="question">

  <h3>3. Which planet is known as the Red Planet?</h3>

  <ul class="options">

    <li><label><input type="radio" name="q3" value="1"> Mars</label></li>

    <li><label><input type="radio" name="q3" value="0"> Venus</label></li>

    <li><label><input type="radio" name="q3" value="0"> Jupiter</label></li>

    <li><label><input type="radio" name="q3" value="0"> Saturn</label></li>

  </ul>

</div>

<button type="button" id="submitQuiz">Submit</button>

</form>

<div class="result" id="result"></div>

</div>

<script>

  document.getElementById("submitQuiz").addEventListener("click", () => {

  const form = document.getElementById("quizForm");

  const resultDiv = document.getElementById("result");

  const formData = new FormData(form);

  let score = 0;

  // Calculate score
```

```
for (let [key, value] of formData.entries()) {  
    score += parseInt(value, 10);  
}  
  
// Display result  
const totalQuestions = 3;  
if (score === totalQuestions) {  
    resultDiv.textContent = Perfect! You scored ${score}/${totalQuestions}.;  
    resultDiv.className = "result success";  
} else if (score >= totalQuestions / 2) {  
    resultDiv.textContent = Good job! You scored ${score}/${totalQuestions}.;  
    resultDiv.className = "result success";  
} else {  
    resultDiv.textContent = Better luck next time. You scored ${score}/${totalQuestions}.;  
    resultDiv.className = "result failure";  
}  
});  
</script>  
</body>  
</html>
```

**OUTPUT**

## Interactive Quiz

**1. What is the capital of France?**

☐ Berlin

☐ Paris

☐ Madrid

☐ Rome

**2. What is  $2 + 2$ ?**

☐ 3

☐ 4

☐ 5

☐ 6

**3. Which planet is known as the Red Planet?**

☐ Mars

☐ Venus

☐ Jupiter

☐ Saturn

**Submit**

**RESULT**

Thus, the quiz has been created successfully.

**Ex.No : 4****SIMPLE ANIMATION USING HTML, CSS, JAVASCRIPT****AIM:**

Create a simple animation using HTML, CSS and JavaScript.

- Use CSS to define the animation's properties, such as duration and timing.
- Use JavaScript to trigger the animation in response to user input or other events

**CODE**

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Simple Animation</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      display: flex;

      flex-direction: column;

      align-items: center;

      justify-content: center;

      height: 100vh;

      margin: 0;

      background-color: #f0f0f0;

    }

    .box {

      width: 50px;
```



```
height: 50px;  
background-color: #3498db;  
position: relative;  
}
```

```
@keyframes moveRight {  
  from {  
    transform: translateX(0);  
  }  
  to {  
    transform: translateX(300px);  
  }  
}
```

```
button {  
  margin-top: 20px;  
  padding: 10px 20px;  
  font-size: 16px;  
  cursor: pointer;  
  background-color: #2ecc71;  
  color: white;  
  border: none;  
  border-radius: 5px;  
}
```

```
button:hover {
```

```
    background-color: #27ae60;
  }
</style>
</head>
<body>
  <div class="box" id="box"></div>
  <button id="animateButton">Animate</button>

  <script>
    document.getElementById("animateButton").addEventListener("click", () => {
      const box = document.getElementById("box");
      box.style.animation = "moveRight 2s ease-in-out";

      // Remove animation after it ends to allow retriggering
      box.addEventListener("animationend", () => {
        box.style.animation = "";
      });
    });
  </script>
</body>
</html>
```

## OUTPUT



## RESULT

Thus, the animation has been created successfully.

Ex.No : 5	GIT & GITHUB OPERATIONS

**AIM**

Write the git and GitHub commands to perform the operations for

- Setting up a local repository,
- Managing multiple branches,
- CRUD with Shell scripting.

**GIT COMMANDS****Set up a local repository.****1. Setup the Working Directory for a New Project:**

```
MINGW64:/i/pro  
  
Lenovo@SRM MINGW64 /i  
$ mkdir pro  
  
Lenovo@SRM MINGW64 /i  
$ cd pro
```

**2. Create a new file in Git & Initialize a new Git Repository (git init)**

```
Lenovo@SRM MINGW64 /i/pro  
$ git init  
Initialized empty Git repository in I:/pro/.git/
```

### 3. Staging File Changes for Tracking (git add <file>...)

```
Lenovo@SRM MINGW64 /i/pro (master)
$ touch index.html

Lenovo@SRM MINGW64 /i/pro (master)
$ git add index.html
```

### 4. Committing File Changes (git commit)

```
Lenovo@SRM MINGW64 /i/pro (master)
$ git commit -m "hello this is an git demo"
[master (root-commit) 96887cd] hello this is an git demo
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
```

### 5. Viewing the Commit Data (git log)

```

Lenovo@SRM MINGW64 /i/pro (master)
$ git log
commit 96887cda9e8dbc877bf76222aaf1c6886148179f (HEAD -> master)
Author: pikachu <msram2006@gmail.com>
Date: Sat Jan 4 17:29:38 2025 +0530

hello this is an git demo

```

## 6. Viewing the difference between committed file and staged file.

- We can inspect all the unstaged changes using "git diff" command (or "git diff <file>" for the specified file). It shows the file changes in the working tree since the last commit:

```

Lenovo@SRM MINGW64 /i/pro (master)
$ vi index.html

Lenovo@SRM MINGW64 /i/pro (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   index.html

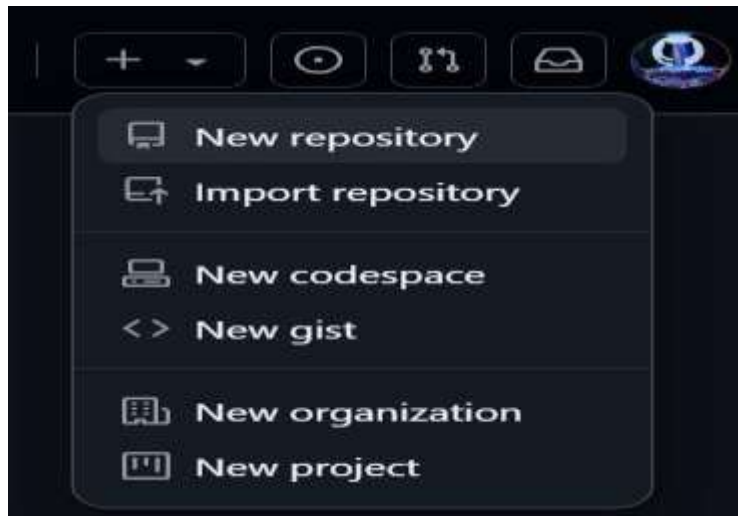
no changes added to commit (use "git add" and/or "git commit -a")

Lenovo@SRM MINGW64 /i/pro (master)
$ git diff index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it
diff --git a/index.html b/index.html
index e69de29..3fa4443 100644
--- a/index.html
+++ b/index.html
@@ -0,0 +1 @@
+this is an edit to this file

```

## 7. Setting up Remote Repository

1. Sign up for a GIT host, such as GitHub <https://github.com/signup/free>
2. Login to the GIT host.
3. Create an empty remote repository in GitHub where we will push our code.
4. Click on the plus sign in the top right corner and choose New repository.

A screenshot of the 'Create a new repository' form in GitHub. The form has a dark theme. At the top, it says 'Create a new repository' and 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, it says 'Required fields are marked with an asterisk (\*)'. The form has two main sections: 'Owner \*' and 'Repository name \*'. The 'Owner' field has a dropdown menu showing 'sriram687' with a plus icon. The 'Repository name' field has a text input with 'Add\_ass' and a green checkmark below it saying 'Add\_ass is available.'. Below these fields, there is a note: 'Great repository names are short and memorable. Need inspiration? How about [redesigned-spoon](#) ?'. The 'Description (optional)' field has a text input with 'git usage for ADD assignment'. At the bottom, there are two radio buttons for visibility: 'Public' (selected) and 'Private'. The 'Public' option has a note: 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option has a note: 'You choose who can see and commit to this repository.'

8. You can list all the remote names and their corresponding URLs via "git remote -v", for example.

```

Lenovo@SRM MINGW64 /i/pro (master)
$ git remote add origin https://github.com/sriram687/Add_ass.git

Lenovo@SRM MINGW64 /i/pro (master)
$ ^[[200~git remote -v
bash: $'\E[200~git': command not found

Lenovo@SRM MINGW64 /i/pro (master)
$ git remote add origin https://github.com/sriram687/Add_ass.git
error: remote origin already exists.

Lenovo@SRM MINGW64 /i/pro (master)
$ git remote -v
origin https://github.com/sriram687/Add_ass.git (fetch)
origin https://github.com/sriram687/Add_ass.git (push)

```

9. Push the commits from the local repo to the remote repo via "git push -u <remote-name> <local-branch-name>".

```

Lenovo@SRM MINGW64 /i/pro (master)
$ git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 457 bytes | 228.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/sriram687/Add_ass.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

```

10. On your local system, make some changes (e.g., on "index.html"); stage and commit the changes on the local repo; and push it to the remote. This is known as the "Edit/Stage/Commit/Push" cycle.



```

Lenovo@SRM MINGW64 /i/pro (master)
$ vi index.html

Lenovo@SRM MINGW64 /i/pro (master)
$ git commit -m "edited once more"
[master 298f6aa] edited once more
1 file changed, 1 insertion(+), 2 deletions(-)

Lenovo@SRM MINGW64 /i/pro (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 277 bytes | 92.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/sriram687/Add_ass.git
   ba31bda..298f6aa  master -> master
branch 'master' set up to track 'origin/master'.

```

### Cloning a Project from a Remote Repo (git clone <remote-url>)

11. Anyone having read access to your remote repo can clone your project. You can also clone any project in any public remote repo. The "git clone <remote-url>" initializes a local repo and copies all files into the working tree. You can find the URL of a remote repo from the Git host.

### Manage multiple branches.

- Branching allows you and your team members to work on different aspects of the software concurrently (on so-called feature branches), and merge into the master branch as and when they are complete.

#### 1. Creating a new Branch (git branch <branch-name>)

```

Lenovo@SRM MINGW64 /i/pro (master)
$ git branch
* master

Lenovo@SRM MINGW64 /i/pro (master)
$ git branch new

```

t

## 2. Switching to a Branch (git checkout <branch-name>)

Git uses a special pointer called HEAD to keep track of the branch that you are working on. The "git branch <branch-name>" command simply create a branch, but does not switch to the new branch. To switch to a branch, use "git checkout <branch-name>" command.

```
Lenovo@SRM MINGW64 /i/pro (master)
$ git checkout new
Switched to branch 'new'
M      index.html
```

## 3. Merging Two Branches (git merge <branch-name>)

To merge two branches, say master and devel, check out the first branch, e.g, master, (via "git checkout <branch-name>") and merge with another branch, e.g., devel, via command "git merge <branch-name>".

```
Lenovo@SRM MINGW64 /i/pro (master)
$ git merge new
Already up to date.

Lenovo@SRM MINGW64 /i/pro (master)
$ git branch --merged
* master
  new
```

## 4. Deleting a Merged Branch (git branch -d <branch-name>)

The merged branch (e.g., devel) is no longer needed. You can delete it via "git branch -d <branch-name>".

```
Lenovo@SRM MINGW64 /i/pro (master)
$ git branch -d new
Deleted branch new (was 298f6aa).
```

## CRUD operations using Shell scripting

```
MINGW64:/i/pro
$ echo "Creating a new entry..."
Creating a new entry...

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "New entry: Item 1" >> data.txt

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Item 1 has been added."
Item 1 has been added.

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Reading the file contents..."
Reading the file contents...

Lenovo@SRM MINGW64 /i/pro (master)
$ cat data.txt
New entry: Item 1

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Updating entry..."
Updating entry...

Lenovo@SRM MINGW64 /i/pro (master)
$ sed -i 's/Item 1/Updated Item 1/' data.txt

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Item 1 has been updated."
Item 1 has been updated.

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Reading the updated file..."
Reading the updated file...

Lenovo@SRM MINGW64 /i/pro (master)
$ cat data.txt
New entry: Updated Item 1

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Deleting entry..."
Deleting entry...

Lenovo@SRM MINGW64 /i/pro (master)
$ sed -i '/Updated Item 1/d' data.txt

Lenovo@SRM MINGW64 /i/pro (master)
$ echo "Item has been deleted."
Item has been deleted.
```

### RESULT:

Thus, the git and GitHub commands are executed successfully and verified

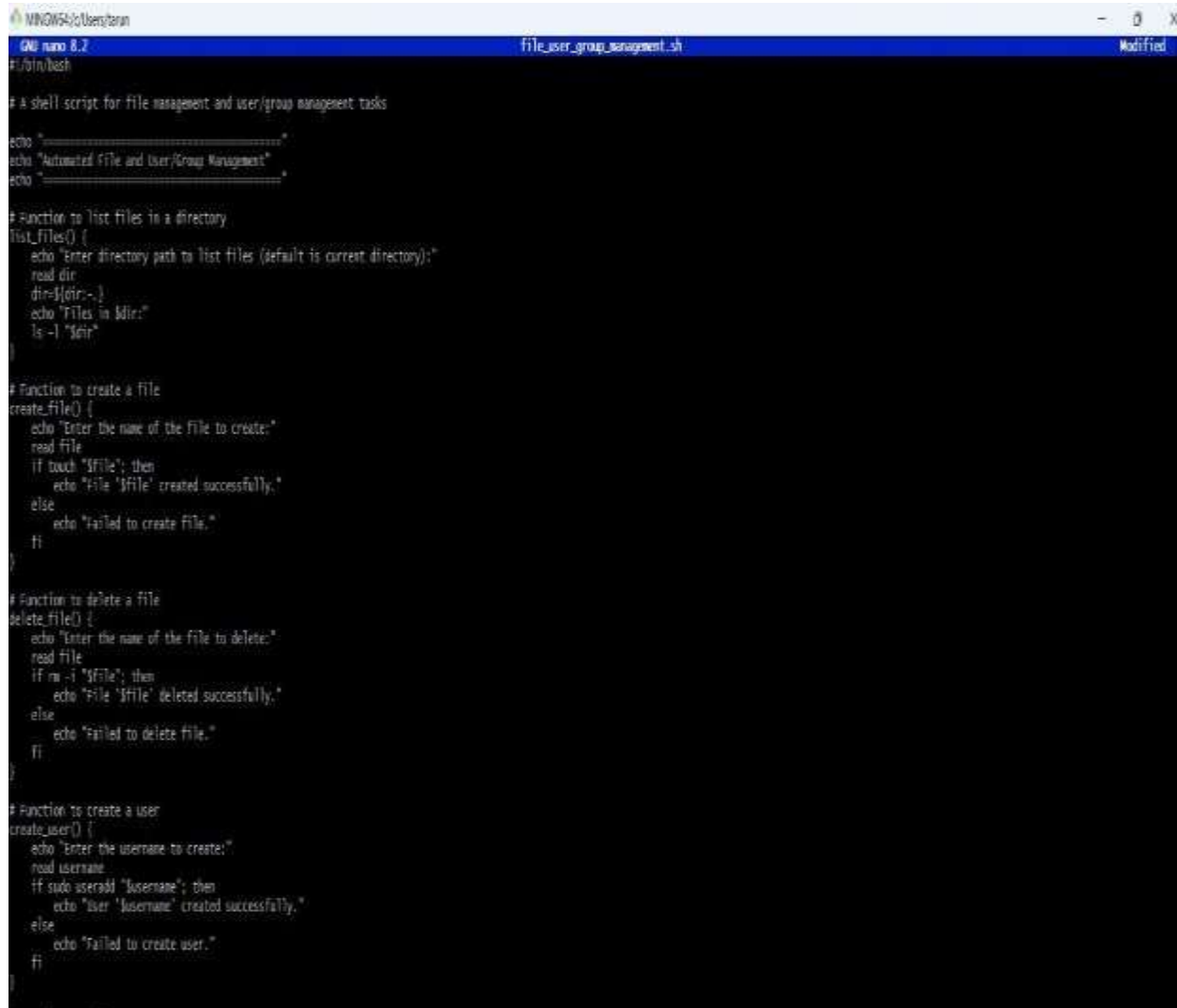
Ex. No: 6

## FILE MANAGEMENT USING SHELL SCRIPTING

## AIM

Create a shell script that can automate file management tasks such as processing, listing, creation and handling of files and user and group management tasks

```
tarun@LAPTOP-5EEMLUJ7 MINGW64 ~ (master)
$ nano file_user_group_management.sh
```



```

GNU nano 8.2 file_user_group_management.sh
# a shell script for file management and user/group management tasks

echo "=====
echo "Automated File and User/Group Management"
echo "=====

# function to list files in a directory
list_files() {
    echo "Enter directory path to list files (default is current directory):"
    read dir
    dir=${dir:-.}
    echo "Files in $dir:"
    ls -l "$dir"
}

# function to create a file
create_file() {
    echo "Enter the name of the file to create:"
    read file
    if touch "$file"; then
        echo "File '$file' created successfully."
    else
        echo "Failed to create file."
    fi
}

# function to delete a file
delete_file() {
    echo "Enter the name of the file to delete:"
    read file
    if rm -i "$file"; then
        echo "File '$file' deleted successfully."
    else
        echo "Failed to delete file."
    fi
}

# function to create a user
create_user() {
    echo "Enter the username to create:"
    read username
    if sudo useradd "$username"; then
        echo "User '$username' created successfully."
    else
        echo "Failed to create user."
    fi
}

```

```

# Function to delete a user
delete_user() {
    echo "Enter the username to delete:"
    read username
    if sudo whoami < /dev/null; then
        echo "User '$username' deleted successfully."
    else
        echo "Failed to delete user."
    fi
}

# Function to create a group
create_group() {
    echo "Enter the group name to create:"
    read groupname
    if sudo groupadd "$groupname"; then
        echo "Group '$groupname' created successfully."
    else
        echo "Failed to create group."
    fi
}

# Function to delete a group
delete_group() {
    echo "Enter the group name to delete:"
    read groupname
    if sudo groupdel "$groupname"; then
        echo "Group '$groupname' deleted successfully."
    else
        echo "Failed to delete group."
    fi
}

# Display menu
display_menu() {
    echo "
Please select an option:
1. List files in a directory
2. Create a file
3. Delete a file
4. Create a user
5. Delete a user
6. Create a group
7. Delete a group
8. Exit"
}

```

```

tarun@LAPTOP-5EEMLUJ7 MINGW64 ~ (master)
$ chmod +x file_user_group_management.sh

```

```

skgra@Sasitharan MINGW64 ~ (main)
$ ^[[200~Creating directory 'DemoFiles'...
Files 'file1.txt', 'file2.txt', and 'file3.txt' created.
bash: $'\E[[200~Creating': command not found
Adding content to 'file1.txt'...
Adding content to 'file2.txt'...
Displaying contents of files:
This is the content of file1.
This is the content of file2.
Renaming 'file1.txt' to 'renamed_file1.txt'...
File renamed successfully.
Creating subdirectory 'SubDir'...
Moving 'file2.txt' and 'file3.txt' to 'SubDir'...
Files moved successfully.
Deleting 'renamed_file1.txt'...
'renamed_file1.txt' deleted.
Final directory structure:
.:
SubDir

./SubDir:
file2.txt
file3.txt

```

## RESULT

Thus, the scripts are executed successfully and verified.

<b>Ex.No : 7a</b>	<b>SIMPLE CALCULATOR -MIT APP INVENTOR</b>

**AIM**

To create **Simple Calculator** mobile application using MIT APP INVENTOR

**PROCEDURE****1. Access MIT App Inventor:**

- Open your web browser and go to the MIT App Inventor website:  
[<https://appinventor.mit.edu/>](<https://appinventor.mit.edu/>).

- Sign in with your Google account.

**2. Create a New Project:**

- Click on "Start New Project."

- Give your project a name, and choose a project template if desired.

**3. Design the User Interface:**

- On the left side, you'll see the "Palette" containing different components. Drag and drop the following components into the designer window:

- `TextBox` (for input and display)
- `Button` (for numbers and operations)
- `Label` (optional, for displaying the operation)

**4. Set Component Properties:**

- Click on each component in the designer window, and in the properties window, set their properties such as `Hint` for TextBox and `Text` for Button.

**5. Add Event Handlers:**

- Click on the "Blocks" button to switch to the Blocks Editor.
- Use the blocks to define the app's behavior. For example:
  - For each number button, create an event handler that appends the corresponding number to the TextBox.
  - For operation buttons (+, -, \*, /), create event handlers to perform calculations and update the TextBox.



- You can use additional variables and logic to handle the calculator's functionality.

## 6. Implement the Calculator Logic:

- Use the math blocks available in MIT App Inventor to perform calculations.
- Create variables to store the current input and result.
- Implement the logic to handle different operations when the corresponding buttons are pressed.

## 7. Testing:

- Connect your Android device to the computer or use the MIT AI2 Companion app to test your calculator in real-time.
- Check for any errors or unexpected behavior and make adjustments in the Blocks Editor.

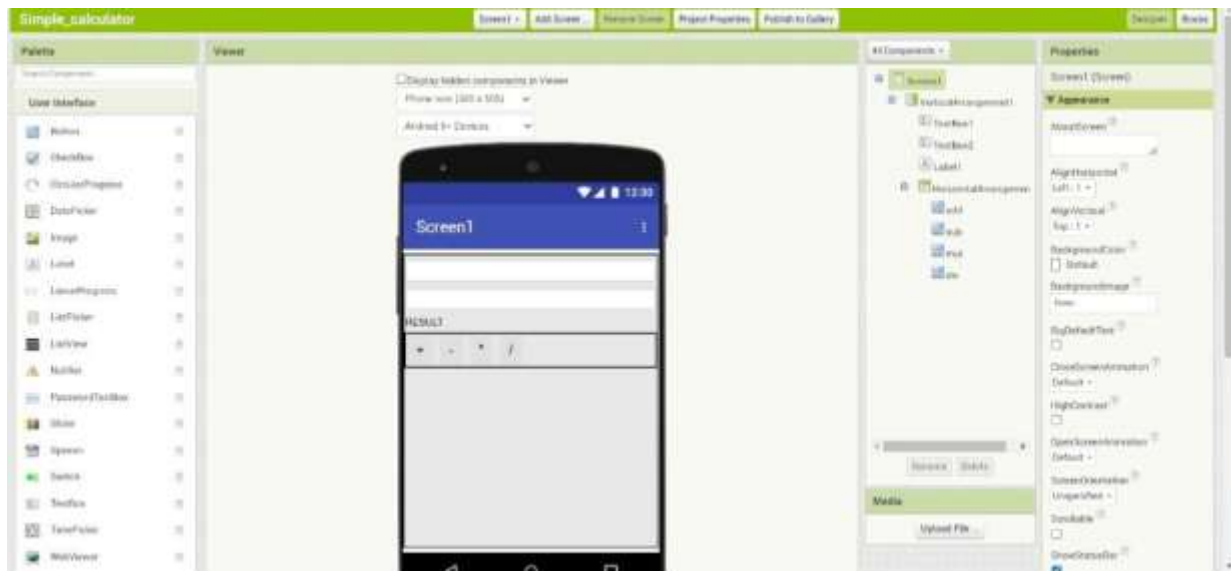
## 8. Build the APK:

- Once you are satisfied with your app, you can build the APK by going to "Build" and selecting "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app, or download the APK and install it on your Android device.

## CODE BLOCKS



## DESIGN & OUTPUT



## RESULT

Thus, the app has been designed successfully.



<b>Ex.No : 7b</b>	<b>STEP COUNTER -MIT APP INVENTOR</b>

**AIM**

To create **Step Counter** mobile application using MIT APP INVENTOR

**PROCEDURE****1. Access MIT App Inventor:**

- Open your web browser and go to the MIT App Inventor website:  
[<https://appinventor.mit.edu/>](<https://appinventor.mit.edu/>).
- Sign in with your Google account.

**2. Create a New Project:**

- Click on "Start New Project."
- Give your project a name, and choose a project template if desired.

**3. Design the User Interface:**

- On the left side, you'll see the "Palette" containing different components. Drag and drop the following components into the designer window:
  - `Label` (for displaying the step count)
  - `Button` (optional, for resetting the step count)
  - `AccelerometerSensor` (for detecting steps)

**4. Set Component Properties:**

- Click on each component in the designer window, and in the properties window, set their properties. For example, set the text of the label to "Step Count: 0."

**5. Add Event Handlers:**

- Click on the "Blocks" button to switch to the Blocks Editor.
- Use the blocks to define the app's behavior. For example:
  - When the screen initializes, you can set initial values and start the accelerometer sensor.
  - Use the accelerometer sensor's `AccelerationChanged` event to detect steps.
  - Update the step count label each time a step is detected.

- Optionally, use a button's click event to reset the step count.

## 6. Implement Step Detection Logic:

- Use variables to store the current step count and implement logic to detect steps based on changes in acceleration.
- You may need to experiment with the accelerometer sensor's sensitivity and thresholds to accurately detect steps.

## 7. Testing:

- Connect your Android device to the computer or use the MIT AI2 Companion app to test your Step Counter in real-time.
- Walk or shake the device to test the step detection logic.

## 8. Build the APK:

- Once you are satisfied with your app, you can build the APK by going to "Build" and selecting "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app or download the APK and install it on your Android device.

## CODE BLOCKS



## DESIGN & OUTPUT



## RESULT

Thus, the mobile application created successfully.

<b>Ex.No : 7c</b>	<b>WEATHER APP -MIT APP INVENTOR</b>

**AIM**

To create a **Weather app** mobile application using MIT APP INVENTOR

**PROCEDURE****1. Get API Key:**

- Sign up for a free account on the OpenWeatherMap website:  
[<https://openweathermap.org/>](<https://openweathermap.org/>).
- Once registered, obtain your API key.

**2. Create a New Project:**

- Open MIT App Inventor and start a new project.

**3. Design the User Interface:**

- Drag and drop the following components into the designer window:
  - `TextBox` for entering the city name or zip code.
  - `Button` to trigger the weather retrieval.
  - `Label` to display the weather information (temperature, description, etc.).

**4. Set Component Properties:**

- Set appropriate properties such as `Hint` for the TextBox and `Text` for the Button.

**5. Add Web Component:**

- Drag the `Web` component from the Palette to the designer. This will be used to make API requests.

**6. Set Web Component Properties:**

- Set the `Web1` component's `Url` property to the OpenWeatherMap API endpoint, including your API key.
- Example URL:  
`https://api.openweathermap.org/data/2.5/weather?q=CityName&appid=YourAPIKey`

**7. Add Event Handlers:**

- Switch to the Blocks Editor and add blocks to handle the Button's click event.
- Use the `Web1` component blocks to make a request to the API when the Button is clicked.
- Parse the JSON response to extract the weather information.

- Update the Label to display the weather details.

## 8. Handle API Response:

- Use blocks from the `Web1` component to handle the API response. You may use blocks like `Web1.GotText` to parse the JSON response.

## 9. Test the App:

- Connect your device or use the MIT AI2 Companion app to test the Weather app.
- Enter a city name or zip code, click the Button, and check if the weather details are displayed.

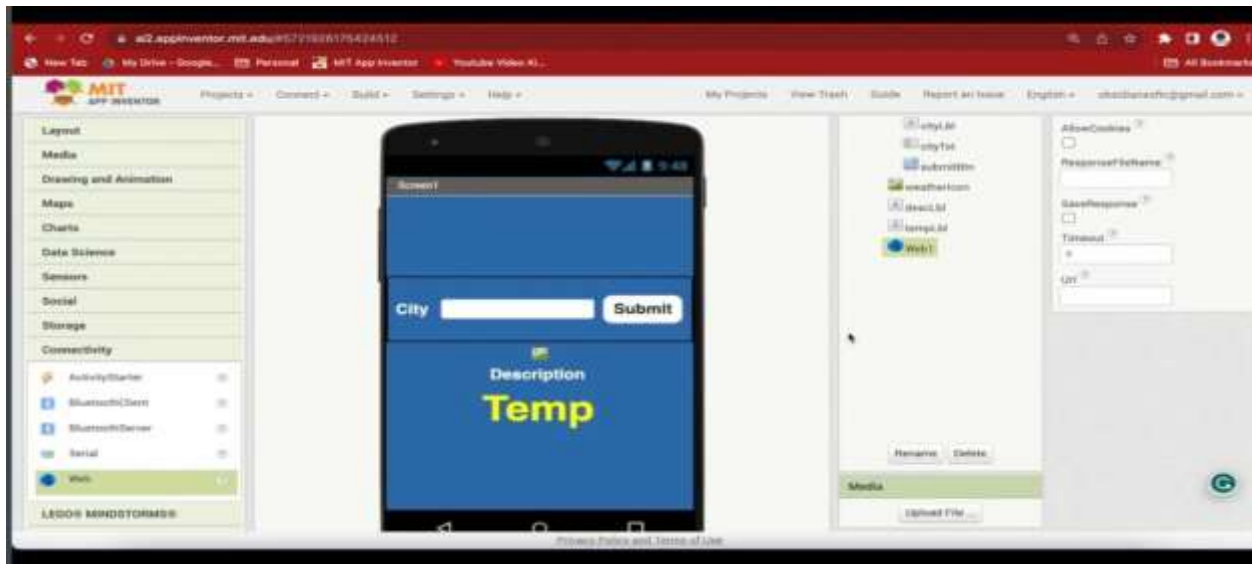
## 10. Build the APK:

- Once satisfied with the app, go to "Build" and select "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app or download the APK and install it on your Android device.

## CODE BLOCKS



## DESIGN & OUTPUT



## RESULT

Thus, the mobile application created successfully.

<b>Ex. No: 7d</b>	<b>TALK TO ME -MIT APP INVENTOR</b>

**AIM**

To create a **Talk to Me** mobile application using MIT APP INVENTOR

**PROCEDURE****1. Access MIT App Inventor:**

- Open your web browser and go to the MIT App Inventor website:  
[<https://appinventor.mit.edu/>](<https://appinventor.mit.edu/>).

- Sign in with your Google account.

**2. Create a New Project:**

- Click on "Start New Project."

- Give your project a name, and choose a project template if desired.

**3. Design the User Interface:**

- On the left side, you'll see the "Palette" containing different components. Drag and drop the following components into the designer window:

- `TextBox` for displaying the spoken text.
- `Button` for triggering speech recognition.
- `TextToSpeech` for converting text to speech (optional).

**4. Set Component Properties:**

- Click on each component in the designer window, and in the properties window, set their properties. For example, set the button text to "Listen."

**5. Add Event Handlers:**

- Click on the "Blocks" button to switch to the Blocks Editor.
- Use the blocks to define the app's behavior. For example:
  - Use the `Button.Click` event to trigger speech recognition.
  - Use the `SpeechRecognizer` component blocks to start listening for speech.

- Handle the `AfterGettingText` event to process the recognized speech and update the TextBox.
- Optionally, use the `TextToSpeech` component to speak a response.

## 6. Implement Speech Recognition:

- Drag the `SpeechRecognizer` component from the Palette to the designer.
- Set the `Language` property to the desired language.
- Use the blocks to start and stop listening for speech, and to handle the recognized text.

## 7. Test the App:

- Connect your Android device to the computer or use the MIT AI2 Companion app to test your "Talk to Me" app.
- Click the button, speak a phrase, and check if the app displays the recognized text.

## 8. Build the APK:

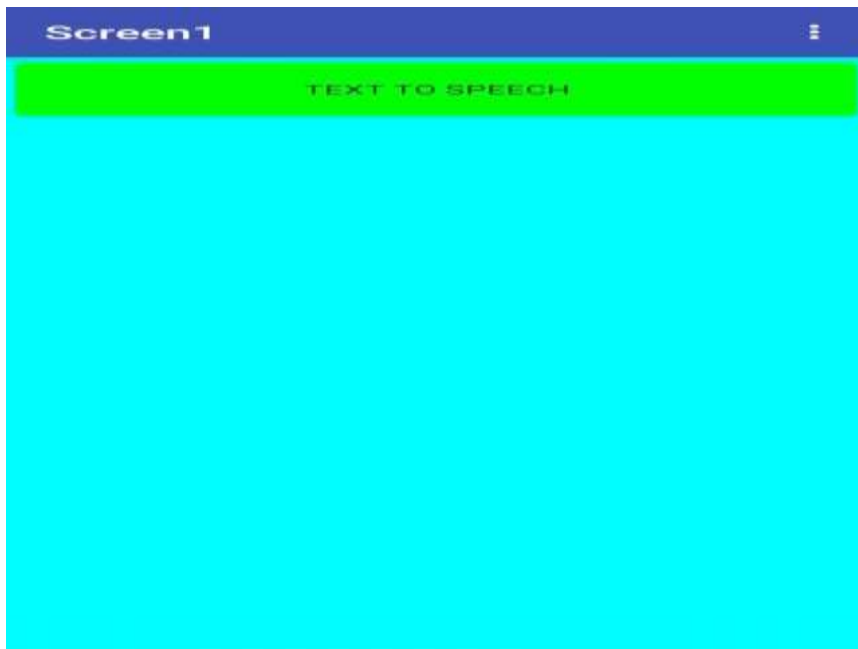
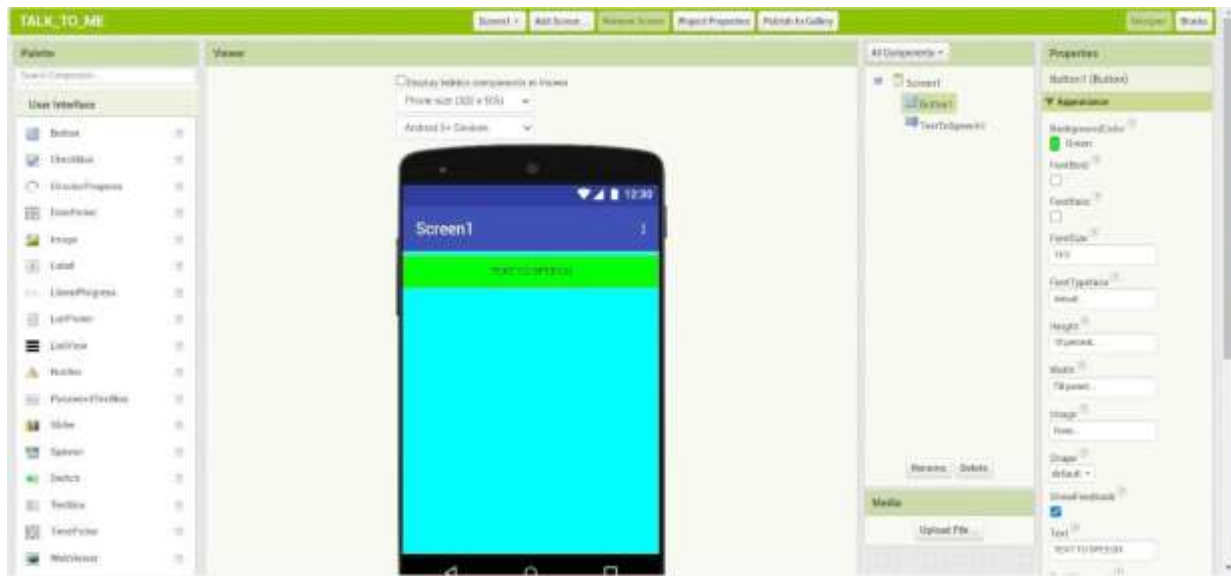
- Once you are satisfied with your app, you can build the APK by going to "Build" and selecting "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app or download the APK and install it on your Android device.

## CODE BLOCKS





## DESIGN & OUTPUT



## RESULT

Thus, the mobile application created successfully.

Ex.No : 7e	<b>TRANSLATION APP -MIT APP INVENTOR</b>

**AIM**

To create a **Translation** mobile application using MIT APP INVENTOR

**PROCEDURE****1. Access MIT App Inventor:**

- Open your web browser and go to the MIT App Inventor website:  
[<https://appinventor.mit.edu/>](<https://appinventor.mit.edu/>).

- Sign in with your Google account.

**2. Create a New Project:**

- Click on "Start New Project."

- Give your project a name, and choose a project template if desired.

**3. Design the User Interface:**

- On the left side, you'll see the "Palette" containing different components. Drag and drop the following components into the designer window:

- `TextBox` for entering text to be translated.

- `Button` to trigger the translation.

- `Label` for displaying the translated text.

**4. Set Component Properties:**

- Click on each component in the designer window, and in the properties window, set their properties. For example, set the button text to "Translate."

**5. Add Web Component:**

- Drag the `Web` component from the Palette to the designer. This will be used to make API requests.

**6. Get API Key for Google Cloud Translation API:**

- Go to the Google Cloud Console:  
[<https://console.cloud.google.com/>](<https://console.cloud.google.com/>).

- Create a new project or use an existing one.

- Enable the "Cloud Translation API" for your project.
- Create credentials and obtain your API key.

### **7. Set Web Component Properties:**

- Set the `Web1` component's `Url` property to the Google Cloud Translation API endpoint, including your API key.
- Example URL: `https://translation.googleapis.com/language/translate/v2?key=YourAPIKey`

### **8. Add Event Handlers:**

- Click on the "Blocks" button to switch to the Blocks Editor.
- Use the blocks to define the app's behavior. For example:
  - Use the `Button.Click` event to trigger the translation.
  - Use the `Web1` component blocks to make a request to the Google Cloud Translation API.
  - Handle the API response to extract and display the translated text in the Label.

### **9. Handle API Response:**

- Use blocks from the `Web1` component to handle the API response. You may use blocks like `Web1.GotText` to parse the JSON response.

### **10. Test the App:**

- Connect your device or use the MIT AI2 Companion app to test the Translation app.
- Enter a text, click the button, and check if the translated text is displayed.

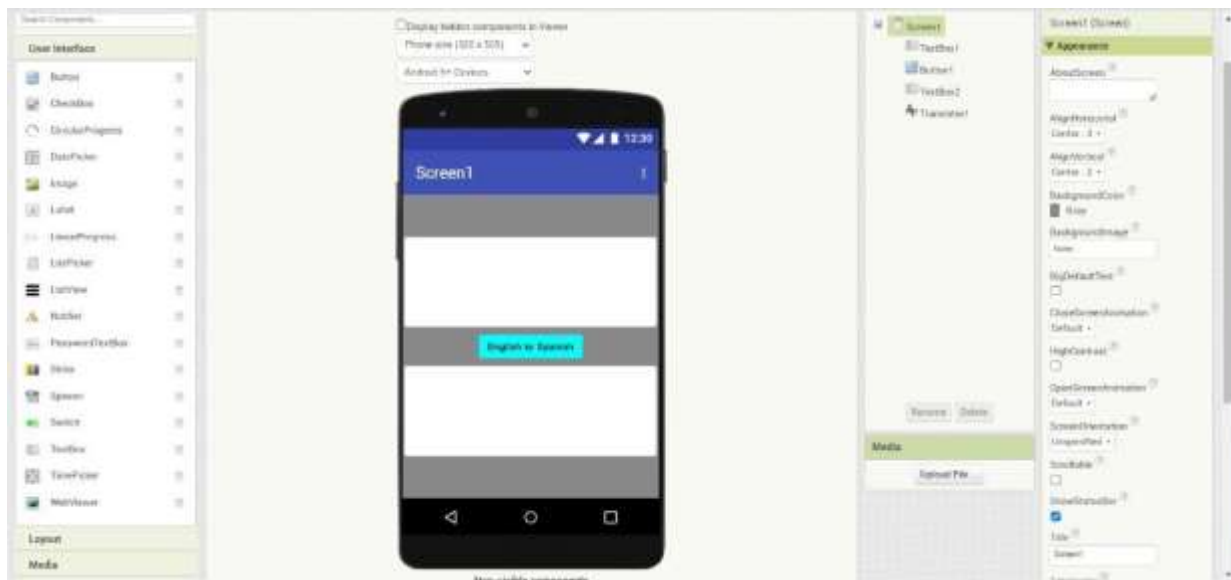
### **11. Build the APK:**

- Once satisfied with the app, go to "Build" and select "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app or download the APK and install it on your Android device.

## CODE BLOCKS



## DESIGN & OUTPUT





Hello friends |



Hola amigos



## RESULT

Thus, the mobile application has been created successfully.

<b>Ex.No : 7f</b>	<b>TO DO LIST -MIT APP INVENTOR</b>

**AIM**

To create **To Do List** mobile application using MIT APP INVENTOR

**1. Access MIT App Inventor:**

- Open your web browser and go to the MIT App Inventor website:  
[<https://appinventor.mit.edu/>](<https://appinventor.mit.edu/>).
- Sign in with your Google account.

**2. Create a New Project:**

- Click on "Start New Project."
- Give your project a name, and choose a project template if desired.

**3. Design the User Interface:**

- On the left side, you'll see the "Palette" containing different components. Drag and drop the following components into the designer window:
  - `TextBox` for entering a new task.
  - `Button` for adding the task to the list.
  - `ListView` for displaying the list of tasks.
  - `Notifier` for displaying alerts.

**4. Set Component Properties:**

- Click on each component in the designer window, and in the properties window, set their properties. For example, set the button text to "Add Task."

**5. Add Event Handlers:**

- Click on the "Blocks" button to switch to the Blocks Editor.
- Use the blocks to define the app's behavior. For example:
  - Use the `Button.Click` event to add the entered task to the list.
  - Use the `ListView` blocks to update the list of tasks.
  - Optionally, use the `Notifier` blocks to display alerts or notifications.

## 6. Implement Task List Logic:

- Use variables to store the list of tasks.
- Implement logic to add tasks to the list and update the `ListView` accordingly.
- You may also add features such as marking tasks as completed or deleting tasks.

## 7. Test the App:

- Connect your Android device to the computer or use the MIT AI2 Companion app to test your To-Do List app.
- Add tasks, mark them as completed, and check if the list is updated.

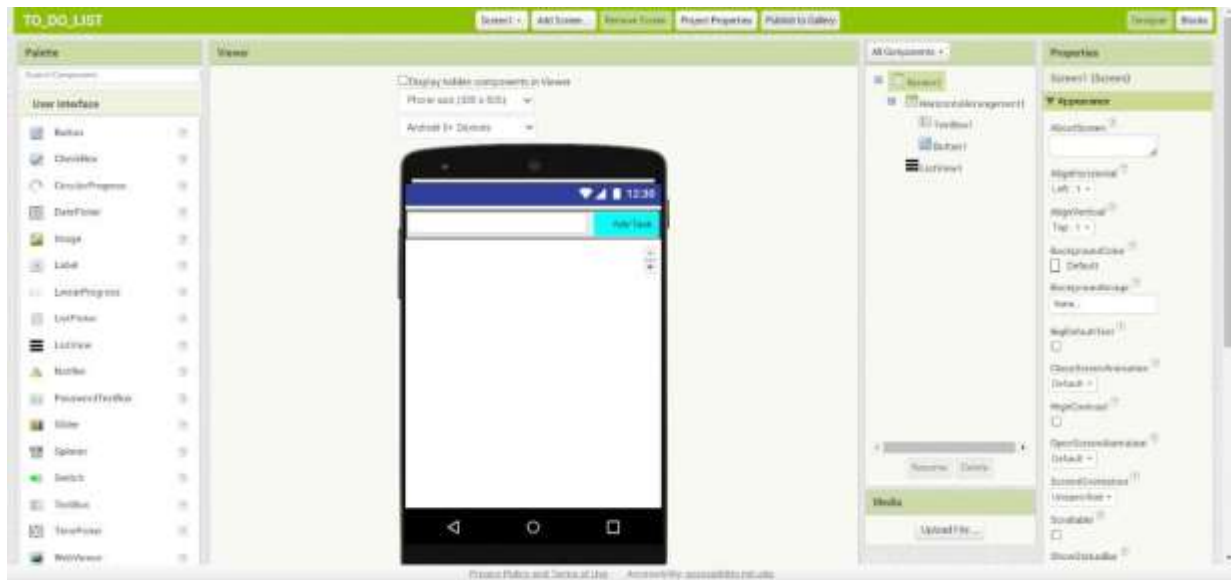
## 8. Build the APK:

- Once you are satisfied with your app, you can build the APK by going to "Build" and selecting "App (provide QR code for .apk)."
- Scan the QR code with the MIT AI2 Companion app or download the APK and install it on your Android device.

## CODE BLOCKS



## DESIGN & OUTPUT



EMP

Add Task

ADD

PSC

TE

CE

EMP

## RESULT

Thus, the mobile application created successfully.



<b>Ex. No: 8</b>	<b>POSTER DESIGN USING CANVA</b>

**AIM:**

To design a poster for any occasion using canva designer tool.

**STEPS:****1. Sign in to Canva:**

- Visit [Canva](https://www.canva.com/).
- Sign in or create a new account.

**2. Choose Poster Dimensions:**

- Click on "Create a design" in the upper right corner.
- Choose "Custom dimensions" and enter the dimensions for your poster (e.g., 18x24 inches).

**3. Select a Template or Start from Scratch:**

- Browse Canva's template library for poster designs or start with a blank canvas.

**4. Choose a Background:**

- Click on the "Background" tab on the left sidebar.
- Select a color, gradient, image, or upload your own to use as the poster background.

**5. Add Text:**

- Click on the "Text" tab on the left sidebar.
- Choose a text style or add your own text box.
- Customize the text by changing the font, size, color, and alignment.

**6. Insert Elements:**

- Use the "Elements" tab to add shapes, lines, icons, or illustrations to enhance your design.
- Drag and drop elements onto your canvas.

**7. Upload Images:**

- If your poster requires images, click on the "Uploads" tab.
- Upload your own images or use Canva's extensive image library.

**8. Arrange and Align Objects:**

- Click on any element, text box, or image to select it.
- Use the toolbar at the top to arrange, align, and distribute elements evenly.

**9. Apply Filters and Effects:**

- Click on any image to reveal the toolbar.
- Use the "Filter" option to apply filters or adjust the transparency.

**10. Add Additional Elements:**

- Continue adding text, elements, or images as needed.
- Experiment with Canva's tools to enhance the visual appeal.

**11. Save and Download:**

- Once satisfied with your poster, click on the "Download" button in the top-right corner.
- Choose the desired file format (e.g., PDF or PNG) and click "Download."

**OUTPUT****RESULT**

Thus, the poster designed successfully using canva tool.

<b>Ex.No : 9</b>	<b>VISUAL INFOGRAPHIC DESIGN USING CANVA</b>

**AIM:**

To create a visual infographic using Canva tools to present data and information.

**STEPS :****1. Sign in to Canva:**

- Visit [Canva](https://www.canva.com/).
- Sign in or create a new account.

**2. Choose Infographic Dimensions:**

- Click on "Create a design" in the upper right corner.
- Scroll down and choose "Infographic."

**3. Select a Template or Start from Scratch:**

- Canva offers a variety of infographic templates. Choose one that suits your purpose or start with a blank canvas.

**4. Add Sections or Blocks:**

- Use the left sidebar to add sections or blocks to your infographic.
- Drag and drop them onto your canvas.

**5. Edit Text:**

- Click on any text box to edit the content.
- Customize the font, size, color, and alignment to suit your design.

**6. Add Icons and Illustrations:**

- Explore the "Elements" tab on the left sidebar.
- Add icons, illustrations, or graphics relevant to your infographic.

**7. Insert Images:**

- Click on the "Elements" tab and select "Images."
- Upload your own images or use Canva's image library.

**8. Use Shapes and Lines:**

- Add shapes or lines to create visual separation and guide the viewer's eyes.
- Customize their colors and sizes.

**9. Apply Colors and Themes:**

- Click on the "Colors" tab on the left sidebar to choose a color palette.
- Canva also offers pre-made themes for a cohesive design.

**10. Add Charts or Graphs:**

- If your information includes data, use the "Charts" tab.
- Input your data, and Canva will generate visually appealing charts.

**11. Customize Layout and Spacing:**

- Click on any element and use the toolbar to customize the layout.
- Adjust spacing and alignment for a clean look.

**12. Add Background:**

- Customize the background by clicking on the "Background" tab.
- Choose a solid color or use an image.

**13. Add Annotations and Labels:**

- Use text boxes or annotations to provide additional information.
- Ensure clarity and conciseness in your labels.

**14. Preview and Adjust:**

- Click on the "Preview" button to see how your infographic looks.
- Make any necessary adjustments for better readability.

**15. Save and Download:**

- Once satisfied, click on the "Download" button in the top-right corner.
- Choose your preferred file format (e.g., PDF or PNG) and download.

## OUTPUT



## RESULT

Thus, the visual infographic design developed successfully using canva tool.

**Ex.No : 10****GAME DEVELOPMENT USING CONSTRUCT 2****AIM:**

To develop a game using construct 2 tool.

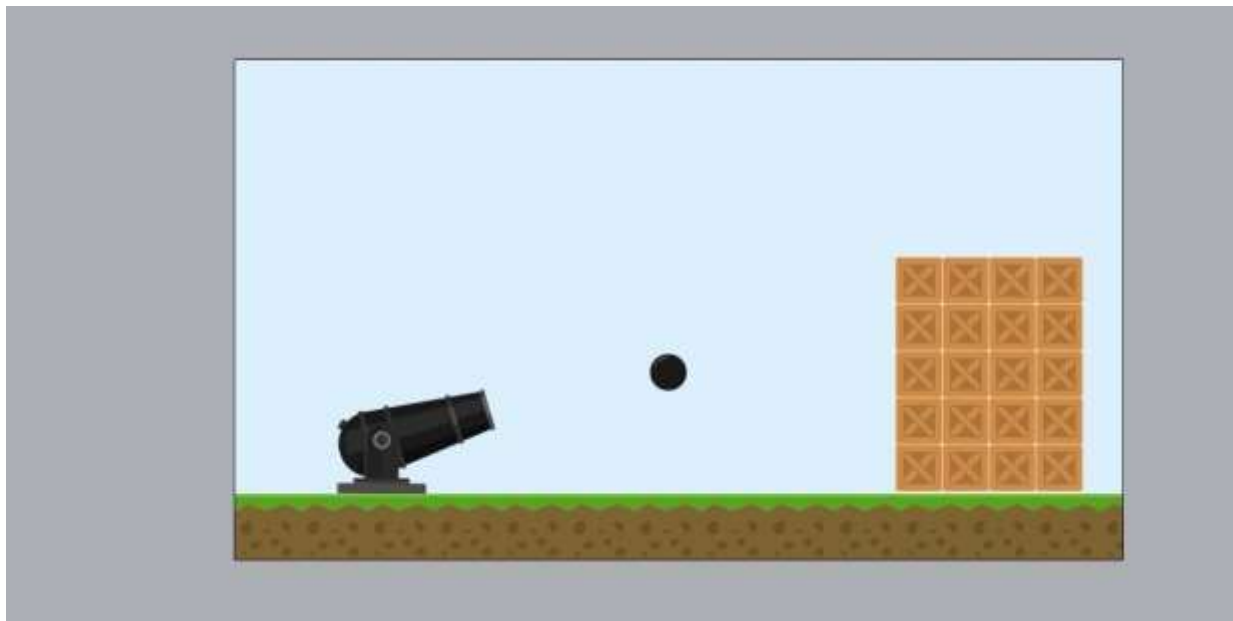
**STEPS :**

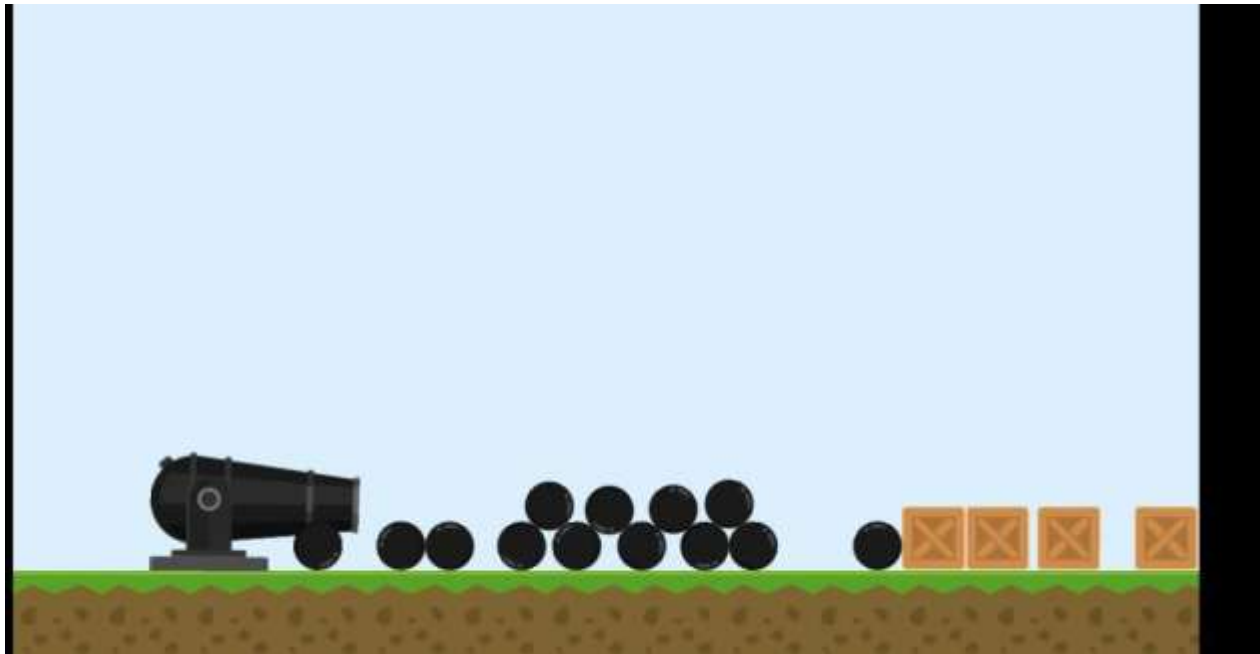
1. Launch Construct 2. Click the File button, and select New. You will see the 'Template or Example' dialog box.
2. The first thing we want is a repeating background tile. The Tiled Background object can do this for us. First, here's your background texture.
3. Now, double click a space in the layout to insert a new object. (Later, if it's full, you can also right-click and select Insert new object.) Once the Insert new object dialog appears, double click the Tiled Background object to insert it.
4. To manage layers, click the Layers tab, which usually is next to the Project bar, You should see Layer 0 in the list (Construct 2 counts starting from zero, since it works better like that in programming). Click the pencil icon and rename it to Background, since it's our background layer.
5. Turn your attention back to the layout. Double click to insert another new object. This time, select the Mouse object, since we'll need mouse input. Do the same again for the Keyboard object.
6. It's time to insert our game objects! Here are your textures - save them all to disk like before.
  - Double click to insert a new object
  - Double click the 'Sprite' object.
  - When the mouse turns to a crosshair, click somewhere in the layout. The tooltip should be 'Main'. (Remember this is the active layout.)
  - The texture editor pops up. Click the open icon, and load one of the four textures.
  - Close the texture editor, saving your changes. You should now see the object in the layout!

## CODE

1	System	On start of layout	ball	Destroy
Add action				
2	System	Every tick	cannon	Set angle toward (Mouse.X, Mouse.Y)
Add action				
3	Mouse	On Left button Clicked	cannon	Spawn ball on layer "ball" (image point T)
			ball	Apply Physics impulse distance(Self.X,Self.Y,Mouse.X,Mouse.Y)/20 at angle cannon.Angle at image point 0
Add action				
Add event				

## OUTPUT





## RESULT

Thus, the above game is developed successfully using construct 2 tool.



<b>Ex. No: 11</b>	<b>MAGE AND VIDEO EDITING USING BLENDER</b>

**AIM:**

To edit the images and video using the blender software tool.

**STEPS:****Image Editing:**

1. Visit Blender.org to download the blender software package and install it in your system.
2. Choose the compositing option in the main menu to select the composite node.
3. Then import your image from the local disk and click the add menu to select the color option.
4. By using color option, the user can create the various nodes such as brightness/contrast, color correction, Hue saturation value, Mix etc.,
5. Use the viewer option in the output menu to view the image editing in background.
6. Use distort option to perform scaling, rotation etc., in the image.
7. Finally, use the render image option in the render menu to see the outcome of the edited image and save the file in your local disk or in a cloud drive.

**Video Editing:**

1. Choose the video editing option in blender and upload the video to be edited from local disk or cloud drive.
2. Set the timeline for video and audio in zero to ensure the synchronization in output video.
3. Click the output properties icon to set the constraints such as format, frame range, output location, audio codec etc.,
4. Finally, use the Render Animation option in the Render menu to generate the edited video and save it to your local disk or in cloud drive.

## OUTPUT



## RESULT

Thus, the image and video were edited successfully using the blender tool.

Ex. No: 12

**SIMPLE ANIMATION USING BLENDER****AIM:**

To animate a short scene or character using Blender animation tools and timeline

**STEPS:**

1. **Open Blender:**
  - a. If you don't have Blender installed, download and install it from the official website: <https://www.blender.org/download/>
2. **Layout Setup:**
  - a. Open Blender and switch to the "Animation" workspace from the top menu.
3. **Import or Create Your Scene:**
  - a. Either import a 3D model or create your scene using Blender's modeling tools.
4. **Set Up Armature (if animating a character):**
  - a. If you're animating a character, create an armature (skeleton) using bones.
  - b. Place bones in key areas like the hips, shoulders, and joints.
5. **Bind Mesh to Armature:**
  - a. Select your 3D model, then shift-select the armature.
  - b. Press **Ctrl + P** to parent the mesh to the armature and choose "With Automatic Weights."
6. **Pose Mode:**
  - a. Switch to Pose Mode for the armature.
  - b. Pose your character by rotating and moving the bones.
7. **Set Keyframes:**
  - a. Move to the first frame on the timeline (default: frame 1).
  - b. Select the bones or objects you want to animate.
  - c. Press **I** and choose the type of keyframe (Location, Rotation, Scale).
8. **Move to Another Frame:**
  - a. Move to a different frame on the timeline.
  - b. Adjust the pose or position of the objects.
  - c. Set another keyframe.
9. **Repeat and Refine:**
  - a. Continue moving to different frames, adjusting poses, and setting keyframes.
  - b. Blender will interpolate between keyframes to create smooth animations.
10. **Timeline and Playback:**
  - a. Use the timeline at the bottom to navigate frames.

- b. Press **Spacebar** to play the animation and check for smoothness.
- 11. **Graph Editor (Optional):**
  - a. For more advanced control, use the Graph Editor to fine-tune animation curves.
- 12. **Camera Animation (Optional):**
  - a. If you have a scene, animate the camera by adding keyframes for its position and rotation.
- 13. **Rendering:**
  - a. Once satisfied, go to the "Render" workspace.
  - b. Set your output settings (resolution, format, etc.).
  - c. Click "Render Animation" to create the final animation.
- 14. **Save and Export:**
  - a. Save your Blender project (**File > Save**).
  - b. Export your animation in a desired format (**File > Export > FBX** for compatibility with other software).
- 15. **External Rendering (Optional):**
  - a. For higher-quality rendering, you might consider using external render engines like Cycles or Eevee.

## OUTPUT



## RESULT

Thus, the animate a short scene or character using Blender animation tools and timeline is successfully completed.