Lab 2

Implementing a Web App Using Git and GitHub for version control

**Introduction**

This lab introduces students to the basics of web applications and version control using Git and GitHub.

**Objectives**

1. Develop a basic understanding of web development.

2. Build a simple web application using Flask.

3. Learn the fundamentals of version control with Git.

4. Get familiar with GitHub for code hosting and collaboration.

**Description**

This lab provides a comprehensive explanation of web development, highlighting key technologies like HTML, CSS, JavaScript, and Flask (a Python-based web framework). Students will explore the process of building a simple Flask web application. The lab also covers the essentials of Git, including how to use it for version control, and introduces GitHub for remote code hosting and team collaboration.

1. **Web development**

**Web development** encompasses the creation and maintenance of websites and web applications. It involves a variety of technologies and disciplines, including HTML, CSS, JavaScript for the front end, and various backend languages and frameworks like Flask & Django (Python), Laravel (PHP), Node (Javascript), .Net(C#), and Spring Boot (Java).

• **Front End vs. Back End**: The front end refers to the parts of a website or application that users interact with directly, while the back end involves the server-side operations that process and manage data.

1. **HTML**

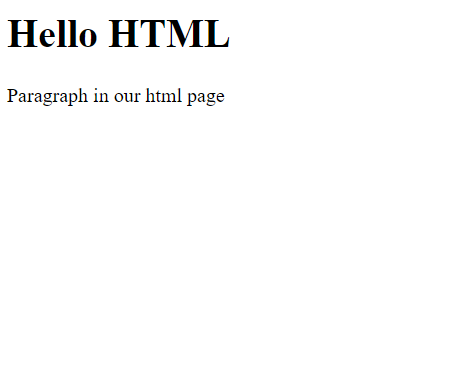
**HTML** stands for Hyper Text Markup Language, consists of a series of elements that tell the browser how to display the content.

**HTML syntax :**

An HTML element is defined by a start tag, some content, and an end tag:

<tagname> Content goes here... </tagname>

**Example** of a basic HTML page:

**<html>**

**<head></head>**

**<body>**

**<h1>Hello HTML</h1>**

**<p>Paragraph in our html page</p>**

**</body>**

**</html>**

* The <html> element is the root element of an HTML page.
* The <head> element contains meta information about the HTML page.
* The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
* The <h1> element defines a large heading.
* The <p> element defines a paragraph.

**Attributes :**

Attributes provide **additional information** about elements

<tagname **attribute=”value”**> Content goes here... </tagname>

**Example** :

<html>

<head></head>

<body>

<h2>IQRA</h2>

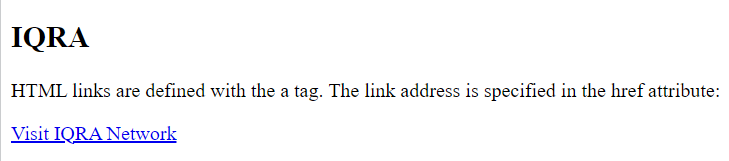
<p>HTML Links are defined with the a tag, the link address is </p>

<a href="https://iqranetwork.com/">Visit IQRA Network</a>

</body>

</html>

Resulting :



* The **<a>** element defines a hyperlink, which is used to link from one page to another.
* **href** attribute is used to define the link.

Any HTML element can have an identifier, either unique using attribute **id** or shared across more than one HTML element using attribute **class.**

**Example :**

<div id="london" class="city">

<h2>London</h2>

<p>London is the capital of England</p>

</div>

**Some HTML tags :**

1. <html> The root element that wraps the entire HTML document.
2. <head> Contains meta-information about the document (like the title, charset, and links to stylesheets or scripts).
3. <title> Sets the title of the web page, which appears in the browser tab.
4. <body> Contains all the visible content of the web page, such as text, images, and links.
5. **<h1> to <h6>** Header tags, used for defining headings on the page, with <h1> being the most important and <h6> the least.
6. <p> Paragraph tag, used to define blocks of text.
7. <a> Anchor tag, used to create hyperlinks to other pages or resources.
8. <img> Image tag, used to embed images into a web page. Requires the src attribute to specify the image file.
9. <ul>**,** <ol>**, and** <li> Unordered list (<ul>), ordered list (<ol>), and list item (<li>) tags, used to create lists.
10. <div> A generic container tag, used to group elements for styling or layout purposes.
11. <span> An inline container tag, used to group text or elements within a larger block for styling purposes.
12. <form> Used to create an HTML form for user input, often containing <input>, <textarea>, and <button> tags.
13. <input> Used to create interactive form elements such as text fields, checkboxes, radio buttons, etc.
14. <button> Used to create clickable buttons.

**HTML forms :**

**HTML forms** are used to collect input from users, such as text, passwords, selections, and more. This input can then be sent to a web server (like Flask!) to be processed.

**Structure of an HTML Form**

A basic HTML form contains these key parts:

1. **Form Tag**: The form starts with a <form> tag, which can include the action attribute to specify where the data should be sent, and the method attribute to define how the data will be sent (GET or POST).
2. **Input Fields**: Inside the form, you add different types of input fields for users to fill in.
3. **Submit Button**: To send the form data, there is usually a "Submit" button.

**Example :**

<html>

<head>

    <title>Simple Form</title>

</head>

<body>

    <h1>Contact Us</h1>

<form action="/submit" method="POST">

        <!-- Text input field for name -->

        <input type="text" id="name" name="name"><br><br>

        <!-- message input field -->

        <input type="message" id="message" name="message"><br><br>

        <input type="submit" value="Submit">

    </form>

</body>

</html>

**Key Elements of a Form:**

1. **<form>**:This tag defines the beginning and end of the form.

**Attributes**

**action:** The URL where the form data will be sent (e.g., /submit).

**method:** The HTTP method for sending the data (usually GET or POST).

<form action="/submit" method="POST">

1. <input>: Used to create a field where the user can type in data.

**Types**:

* **text:** For single-line input like name.
* **email:** For email input with basic validation.
* **password:** For masked password input.
* **submit:** Creates a submit button.

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

<input type="submit">: The submit button sends the form data to the server.

**Form Submission Methods**

**GET**: Sends form data as part of the URL (visible to users).

**Example:** http://example.com/form?name=John&email=john@example.com

Used for simple data requests or when you don't need to hide the information (e.g., search forms).

**POST**: Sends form data in the body of the request (not visible in the URL).

Used when the form data contains sensitive information (like passwords or personal details). More secure than GET for most purposes.

1. **CSS**

**CSS** stands for Cascading Style Sheets, describing how HTML elements are going to be displayed on the screen.

**How to Insert CSS with HTML ?**

1. **External CSS**

With an external style sheet, you can change the look of an entire website by changing just one file, each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

<html>

<head>

<link rel="stylesheet" href="mystyle.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph</p>

</body>

</html>

In the example above we linked our CSS file named “mystyle.css” in our html document.

1. **Internal CSS**

An internal style sheet may be used if one single HTML page has a unique style.The internal style is defined inside the <style> element, inside the head section.

<html>

<head>

<style>

h1{

color:red;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph</p>

</body>

</html>

1. **Inline CSS**

An inline style may be used to apply a unique style for a single element.To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

<html>

<head>

</head>

<body>

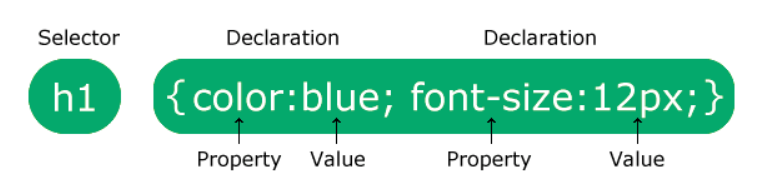
<h1 style="color:blue;text-align: center;">This is a heading</h1>

<p style="color: red;">This is a paragraph</p>

</body>

</html>

**CSS syntax :**



**Selector :** The selector points to the HTML element you want to style, can be tagname as mentioned h1 tag in the above image and can be another pointer to the html elements (#id, .class).

**Declaration** : The declaration block contains one or more declarations separated by semicolons, Each declaration includes a CSS property name and a value, separated by a colon.

In case of selecting element by it’s id we add **#** before the id:

**#elementId** { color:blue; }

In case of selecting elements by their class we add **.** before the class name:

**.elementsClassName** { color:red; }

Some **CSS** attributes with their possible values

| **Attribute name** | **Usage** | **Possible values** |
| --- | --- | --- |
| **color** | Sets the color of text | **Named colors**: red, green,blue etc.  **RGB values**: rgb(255, 0, 0) |
| **font-size** | Defines the size of the font | **Keywords**: small, medium, large  **Pixels**: 14px, 16px, etc. |
| **background-color** | Sets the background color of an element | **Named colors**: red, blue, green, etc.  **RGB values**: rgb(255, 0, 0) |

1. **JavaScript**

**JavaScript** is the programming language of the Web that program the behavior of web pages.

**JavaScript Syntax :**

1. Create variable using keyword **let, const**:

**let x; const z;**

Note that A JavaScript variable can hold any type of data.

**Examples :**

// Numbers:  
let length = 16;  
let weight = 7.5;  
  
// Strings:  
let color = "Yellow";  
let lastName = "Johnson";  
  
// Booleans  
let x = true;  
let y = false;  
  
// Object:  
const person = {firstName:"John", lastName:"Doe"};  
  
// Array object:  
const cars = ["Saab", "Volvo", "BMW"];

1. **Make comments** using **//** for single line and /\* \*/ for multible lines **:**
2. **JavaScript Operators** are used to perform different types of mathematical and logical computations.

**Examples:**

The **Assignment Operator** **=** assigns values

The **Addition Operator** **+** adds values

The **Multiplication Operator** **\*** multiplies values

The **Comparison Operator** **>** compares values

1. **JavaScript Functions :**

A JavaScript function is defined with the **function**keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:  
**(parameter1, parameter2, ...)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

function name(parameter1, parameter2, parameter3) {

// code to be executed

}

**Example** **:**

function sum(number1, number2) {

let sum = number1 + number2;

return sum;

}

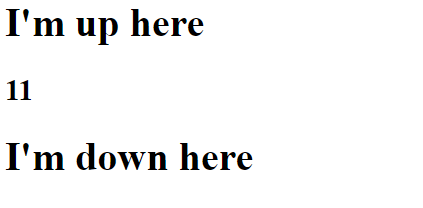
console.log(sum(1,2));

1. **JavaScript Output :**

JavaScript included in HTML using tag **<script>**

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using **innerHTML**.

<html>

<body>

<h1>I'm up here</h1>

<h2 id="demo"></h2>

<h1>I'm down here</h1>

<script>

document.getElementById("demo").innerHTML = 11;

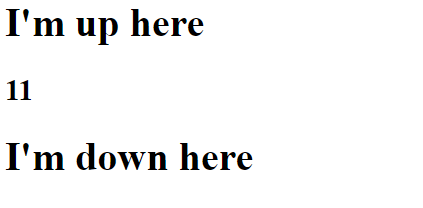
</script>

</body>

</html>

1. Writing into the HTML output using document.write().

<html>

<body>

<h1>I'm up here</h1>

<h2>

<script>

document.write(11);

</script>

</h2>

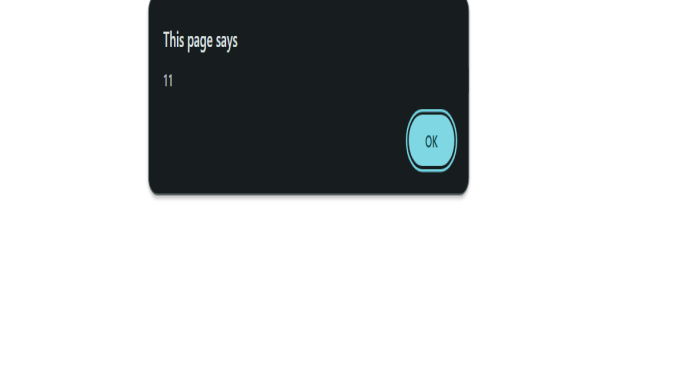
<h1>I'm down here</h1>

</body>

</html>

1. Writing into an alert box, using window.alert().

<html>

<body>

</body>

<script>

window.alert(11);

</script>

</html>

1. Writing into the browser console, using console.log()

<html>

<body>

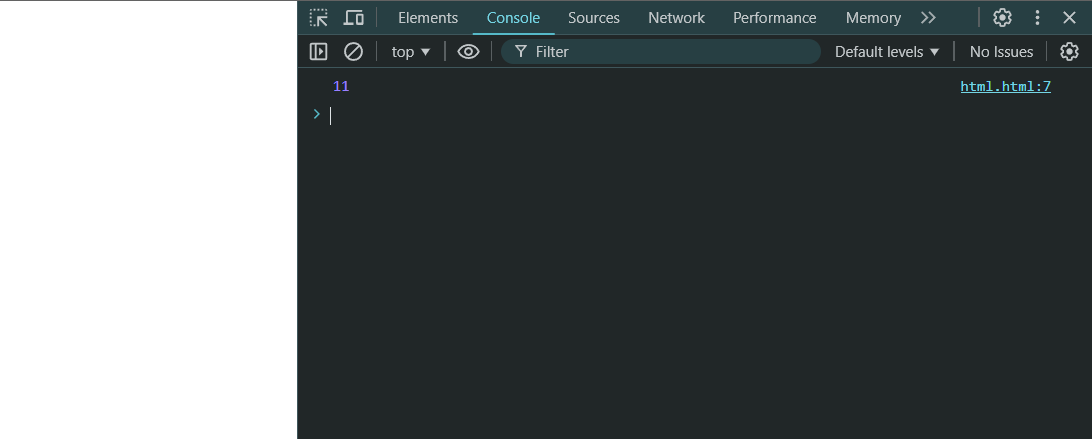
</body>

<script>

console.log(11);

</script>

</html>



**To view console**

press **right-click** -> **inspect** -> **console**

1. **JavaScript with HTML onClick event**

When An HTML element was clicked

<element event='some JavaScript'>

**Example :**

<button onclick="window.alert(Date())">The time is?</button>

And there other events like

**Onmouseover** : The user moves the mouse over an HTML element

**Onmouseout** : The user moves the mouse away from an HTML element

**5- Flask**

**What is Web Framework?**

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

**What is Flask?**

Flask is a lightweight web application framework written in Python. It’s designed to make getting started quick and easy, with the ability to scale up to complex applications.

1. **Installation :**

**The following command installs virtualenv.**

* pip install virtualenv — This command needs administrator privileges.
* Add sudo before pip on Linux/Mac OS. If you are on Windows, log in as Administrator.
* On Ubuntu virtualenv may be installed using its package manager.

Sudo apt-get install virtualenv

**Once installed, new virtual environment is created in a folder.**

mkdir newproj —- cd newproj —- virtualenv venv

To activate corresponding environment

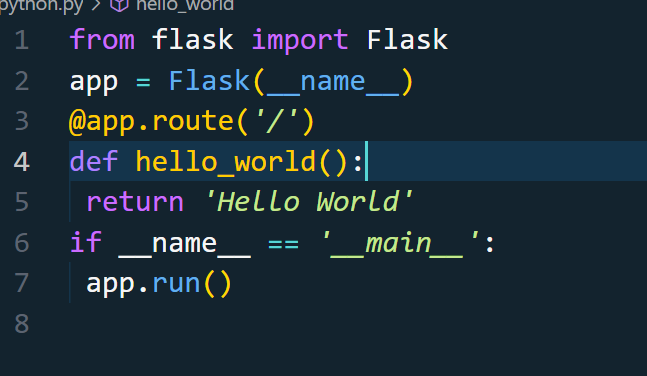
* on Linux/OS X, use the following: venv/bin/activate
* On Windows, following can be used: venv\scripts\activate

**We are now ready to install Flask in this environment.**

pip install Flask

**In order to test Flask installation,**

1. Create new file named **Hello.py** and type the following code inside :



1. Run the following commands in the **newproj directory**:

**venv\Scripts\activate Python Hello.py**

You should see Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

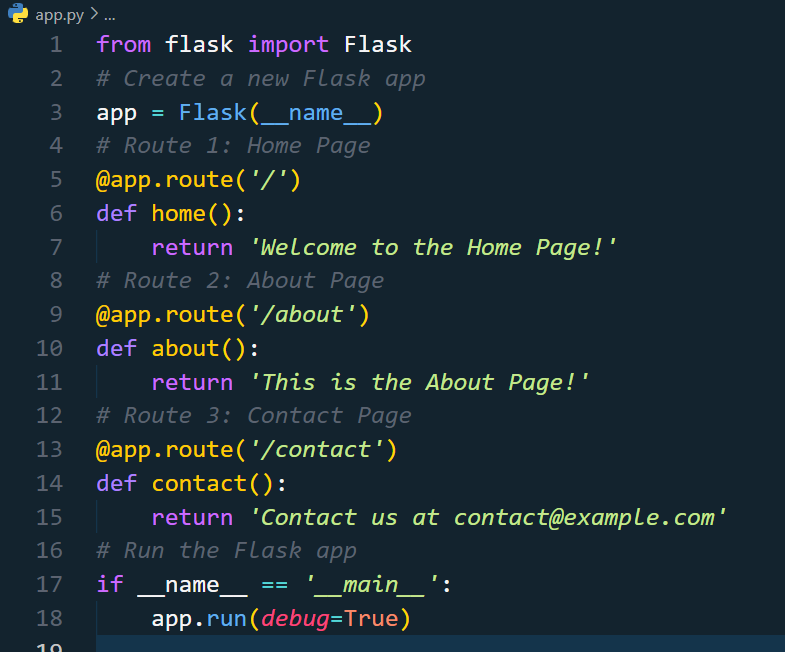
1. **Routing :**

In Flask, **routing** is like giving your web app directions. When someone visits your website, they go to a specific page by entering a URL. Flask uses routing to decide what to show on different pages based on the URL.

Think of it like this: if your website was a house, the **URL** would be the address, and routing tells you which room to go to!

**Example :**

1. you need to install Flask as mentioned above and create a new Python file called app.py.
2. Then inside that file, write this code:



**app.route()**: This tells Flask which URL to listen to. Each **route** is like a different room in your house, and each room shows different content.

**/**: This is the **home page** route (like the front door of a house). When someone visits your website’s homepage, they see what’s in the home() function.

**/about**: This is another route for the "About Page." It’s like telling the person to go to a different room where they can learn more about your website.

**/contact**: This route shows the "Contact Page" with an email address. It’s like a room where you tell people how to reach you.

1. Run the app using these commands in the project directory:

**venv\Scripts\activate Python app.py**

**Done,** Go to the url shown and try the different routes (/about, /contact).

1. **Rendering html files**

you can also open and serve **HTML files** to make your web pages look better and more interactive. Instead of just returning plain text, you can use **HTML templates** to display structured web pages.

**Example:**

1. Create the Project Structure

/templates

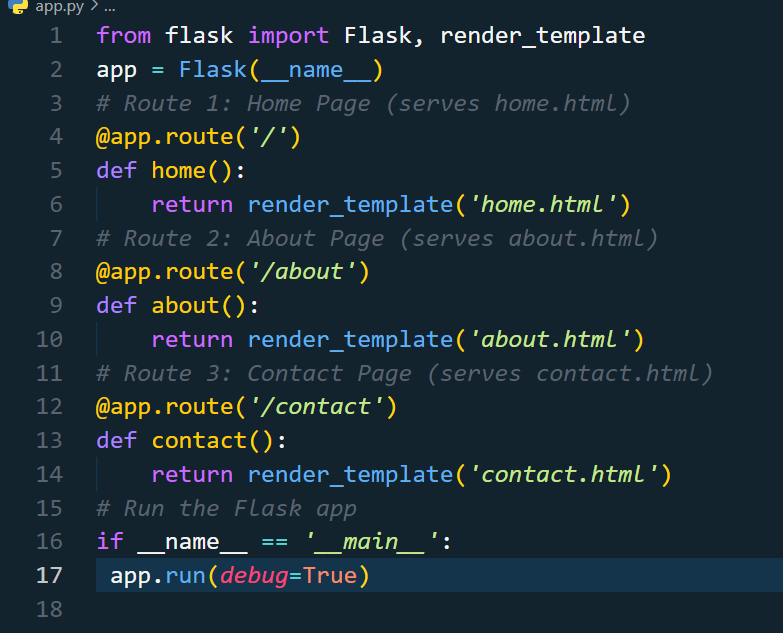
home.html

about.html

contact.html

app.py

1. Create HTML Files and write suitable HTML content for each to be shown.
2. Inside app.py write the following code :



∙**render\_template()**: This is like telling Flask, "Please open this HTML file and show it on the screen." It looks inside the templates folder and shows the content from the HTML file.

1. Run the app using these commands in the project directory:

**venv\Scripts\activate Python app.py**

**Git and Github**

**What is Git?**

Git is a popular version control system. It was created in 2005.

It is used for:

* Tracking code changes
* Tracking who made changes
* Coding collaboration

**What does Git do?**

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

**Why Git?**

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

**What is GitHub?**

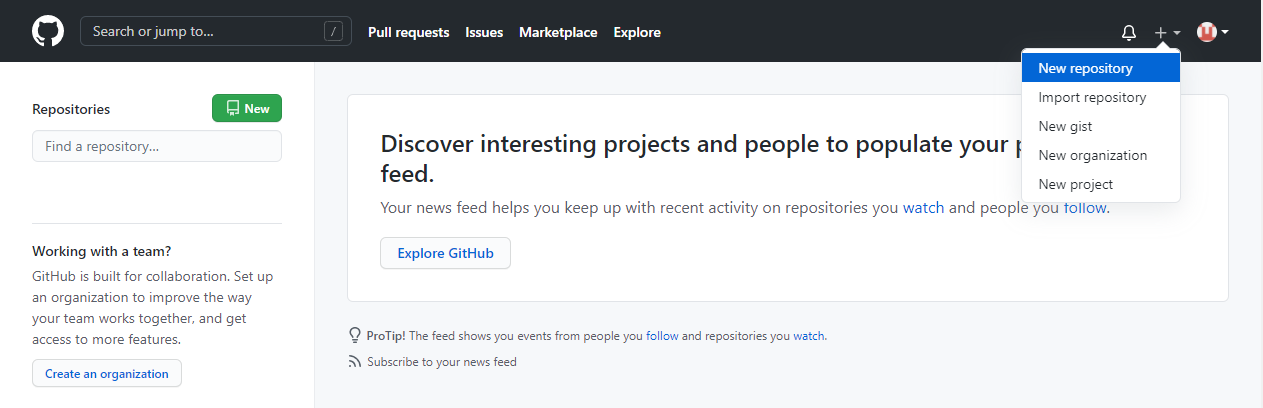
* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
* In this tutorial, we will focus on using Git with GitHub.

**Working with Git**

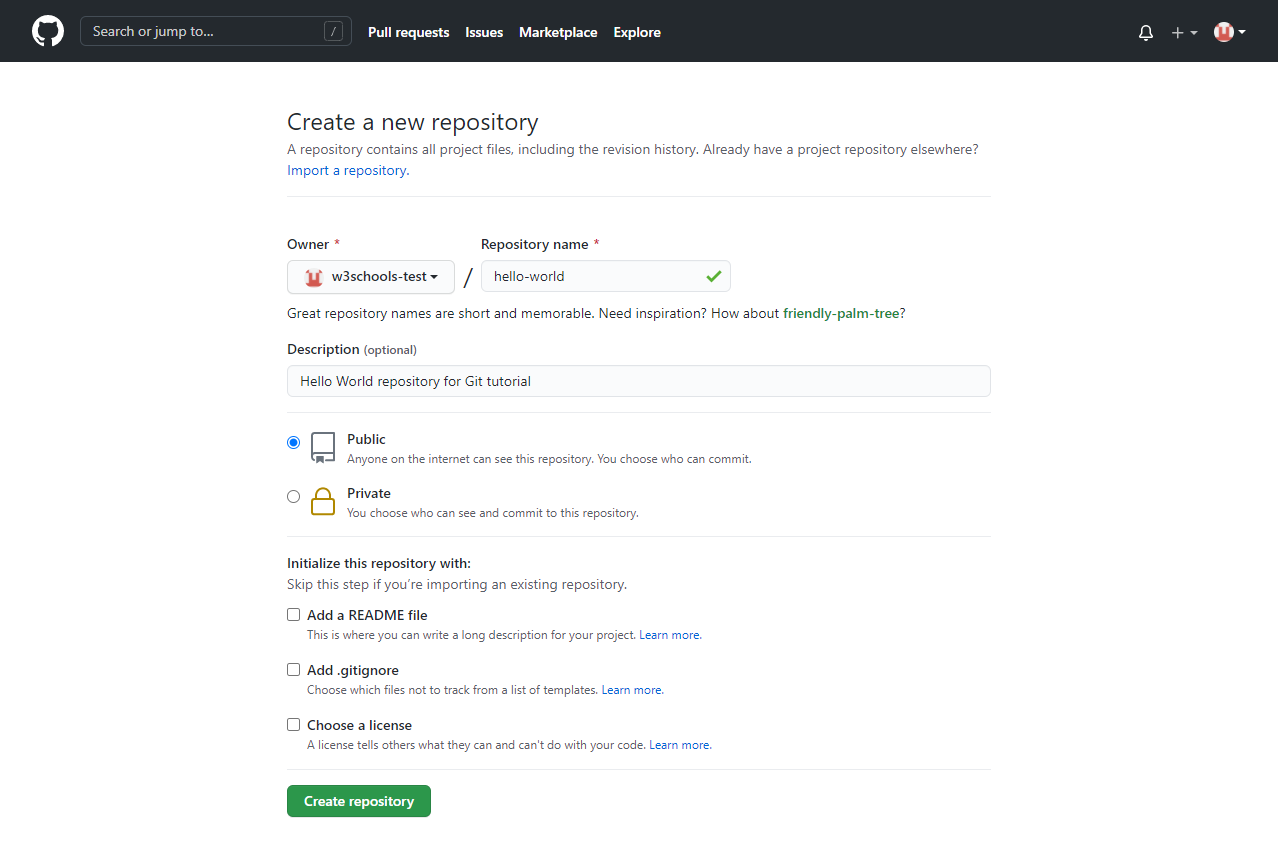
* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

**Get started with GitHub**

1. Go to [GitHub](https://www.github.com/) and sign up for an account:
2. Create a Repository on GitHub



1. Fill in the relevant details:



Choose **Public** (if you want the repo to be viewable for anyone) or Private (if you want to choose who should be able to view the repo). Either way, you will be able to choose who can **contribute** to the repo, Then click "Create repository".

1. Cloning a Repository

Once you've created a repository on GitHub, the next step is to copy it onto your local machine, where you can work on the project. This is called **cloning**.

**git clone https://github.com/username/repo-name.git**

1. Working with Branches

Branches in Git are an essential part of how Git works. A **branch** allows you to work on different parts of a project separately from the main codebase (usually called main or master).

**Create a new branch**: **git checkout -b branch-name**

This creates and switches to a new branch where you can work on newfeatures without affecting the main branch.

**Merge branches**: Once you’re happy with the changes made in a branch, you can merge it back into the main branch.

**git checkout main git merge branch-name**

1. Making Changes Locally

Once you’ve cloned the repository and switched to a branch, you can start making changes to the files on your local machine. After you’ve made your changes:

**Check the status of your changes**: **git status**

This command will show which files have been modified and which are untracked.

**Stage your changes** (prepare them for committing): **git add file-name**

Or to stage all the changed files: **git add .**

**7. Committing Changes**

After staging the files, you need to **commit** them. Committing is like taking a snapshot of your changes, so that Git can keep track of what you’ve done.

**Commit your changes**: **git commit -m "Your commit message"**

Make sure to write clear and concise commit messages that explain what changes you’ve made.

**8. Pushing Changes to GitHub**

Once you have committed your changes, you can upload (or **push**) them to your GitHub repository so that everyone can see them.

**Push your changes**: **git push origin branch-name**

This uploads your changes to the specified branch on GitHub.

**9. Pull Requests**

Once you’ve made changes in your branch and pushed them to GitHub, you can open a **Pull Request (PR)**. A PR is a way of proposing changes to the main branch of the repository.

* **How to open a Pull Request**:
  1. Navigate to your repository on GitHub.
  2. Click on the “Pull Requests” tab.
  3. Click “New Pull Request.”
  4. Select the branch with your changes and the branch you want to merge into (usually main).
  5. Write a description explaining your changes, then submit the pull request.

**10. Collaboration on GitHub**

GitHub allows for collaboration by enabling multiple developers to work together on the same project. Here are some important collaboration workflows:

**Forking a repository**: Forking creates a copy of someone else’s repository under your GitHub account. You can make changes to your fork and submit pull requests to the original repository.

**Collaboration via issues**: Issues allow developers to track bugs, discuss new features, or document project discussions. Use the “Issues” tab in a repository to open new issues or contribute to existing ones.

**11. Pulling Updates from the Remote Repository**

When collaborating, it’s important to keep your local repository up to date with the latest changes from the remote repository.

**Pull the latest changes**: **git pull origin main**

This fetches and integrates any updates from the main branch of the remote repository into your local copy.

**12. Resolving Merge Conflicts**

Sometimes, changes made by different contributors may conflict with each other. Git will notify you when a **merge conflict** occurs, and you’ll need to resolve the conflict manually.

**To resolve a conflict:**

1. Open the conflicting file.
2. Look for the conflict markers (e.g., <<<<, ====, >>>>).
3. Edit the file to fix the conflict.
4. Stage the resolved file and commit your changes.

**Essential Git Commands for GitHub**

Here is a list of some essential Git commands that you’ll frequently use when working with GitHub:

* **git init**: Initialize a new Git repository in a folder.
* **git clone**: Clone a remote repository to your local machine.
* **git status**: Check the status of your changes.
* **git add**: Stage changes for commit.
* **git commit**: Commit the staged changes with a message.
* **git push**: Push the committed changes to the remote repository.
* **git pull**: Fetch and merge changes from the remote repository.
* **git branch**: List, create, or delete branches.
* **git merge**: Merge a branch into your current branch.
* **git log**: View the history of commits.

**Reading (IMP)**

<https://www.w3schools.com/html/>

<https://www.w3schools.com/css/default.asp>

<https://www.w3schools.com/js/default.asp>

**Questions**

**1. What are the differences between the front end and the back end in web development?**

**2. How can you insert CSS into an HTML document?**

**3. What is the purpose of Flask in web development, and how do you install it?**

**4. Explain the steps for creating a Git commit and pushing changes to GitHub.**

**Answer:**

**5. What is a Pull Request (PR) on GitHub, and why is it important?**

**Answer:**

**References**

<https://www.w3schools.com/html/>

<https://www.w3schools.com/css/default.asp>

<https://www.w3schools.com/js/default.asp>

<https://www.tutorialspoint.com/flask/flask_tutorial.pdf>