

# IT3030 (PAF) – Practical Sheet 1

## Version Controlling with Git - I

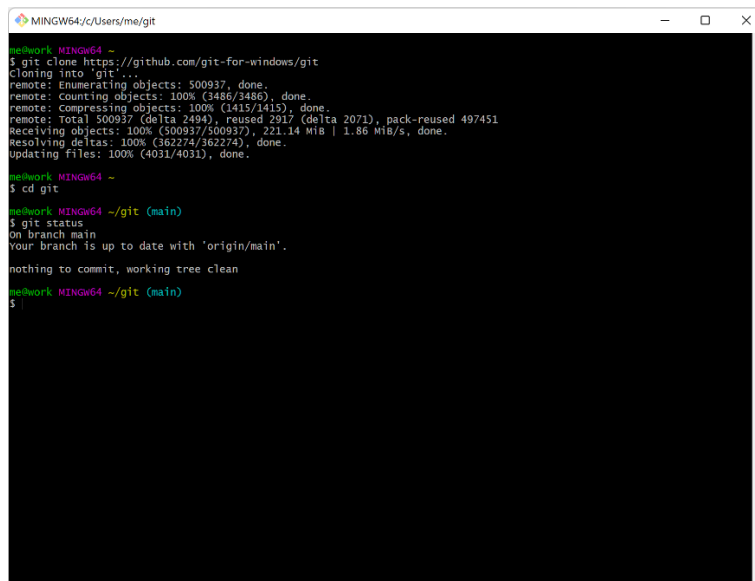
### Introduction

By far, the most widely used modern version control system in the world today is Git. Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source.

You will learn more on the concepts on version controlling and git in PAF lecture sessions. This lab is a gentle introduction to some basic concepts in Git.

### Steps – Part I (Local Repository)

1. Install and configure Git on your computer.
  - For Official Documentation on Git Installation on Windows/ Linux/ Mac follow [this link](#).
2. Once the Installation is complete, open the Git bash if on Windows. On Linux/ Mac, simply open a terminal window. Git bash is a special terminal for Windows only.



```
MINGW64/c:/Users/me/git
$ git clone https://github.com/git-for-windows/git
Cloning into 'git'...
remote: Enumerating objects: 500937, done.
remote: Counting objects: 100% (3486/3486), done.
remote: Compressing objects: 100% (1415/1415), done.
remote: Total 500937 (delta 2494), reused 2917 (delta 2071), pack-reused 497451
Receiving objects: 100% (500937/500937), 221.14 MiB | 1.86 MiB/s, done.
Resolving deltas: 100% (362274/362274), done.
Updating files: 100% (4031/4031), done.
MINGW64/c:/Users/me/git
$ cd git
MINGW64/c:/Users/me/git
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
MINGW64/c:/Users/me/git
$
```

Figure 1: Git Bash on Windows

3. The next step in adding Version Control with git to a project is to initialize the repository.
  - Create a new folder for your repository on your machine.
  - Then inside this new folder, to create a new repo, you'll use the git init command. git init is a one-time command you use during the initial setup of a new repo. Executing this command will create a new .git subdirectory in your current working directory. This command is used to start a new repository.

Usage: `git init <project directory>`

4. Next, create HelloWorld.html inside the newly initiated git repository.

```
<html>

    <head>

    </head>

    <body>

        <h1>Hello World</h1>

    </body>

</html>
```

5. Next step is to add the created file to the local repository. For this purpose, git add command will be used. This command adds a change in the working directory to the staging area of the local repository.

- Usage: `git add <filename>` (to add a specific file)
- Usage: `git add --all` (to add all added/ modified/ deleted files in the repository)

6. You can use the command `git status` to see the status of the working directory. It is a good habit to periodically check the status of your repository. Execute this command now and observe the output. Try to understand what it means.

7. Finally, once the new/ modified file(s) is/ are added to the staging area, next task to do is to commit them to the local repository. Commit command records or snapshots the file permanently in the version history.

- Usage: `git commit -m "Type in the commit message"`
- It is a good practice to write meaningful commit messages so that you and others who refer to your repository later can understand what changes/ additions you made in this particular commit.
- Once this step is done, your file is now being tracked by Git. Anytime a change happens to this file from now onwards, git will notice and you can effectively keep a check on how the file is modified.

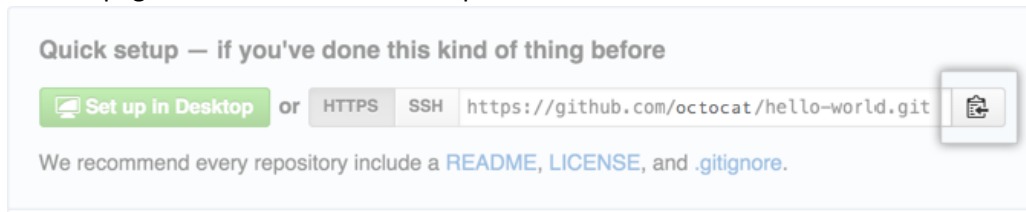
## Steps – Part II (Remote Repository)

Remote repositories are versions of your project that are hosted on the Internet or network somewhere. To be able to collaborate on any Git project, you need to know how to manage your remote repositories. Let's see in this small activity, how to push your local repository to a remote repository so you can collaborate with others as well.

Github is one well known space to host remote repositories. So, we will Github for this example. But you could try the same with other providers such as BitBucket, GitLab and Azure DevOps and the process should be more or less similar.

1. First login or sign-up to Github.

2. Then create an empty repository on Github following instructions on [this documentation](#). (You may create a Public or Private type repository).
3. Then you need to specify that this remote repository on Github is linked to the local repository on your personal computer. To do so, follow the instructions [here](#).
  - a. The remote url for the GitHub Repository you created can be found on that Repository's page itself. Select the HTTPS option for now.



4. Then follow the instructions available [here](#) from step 9. Steps 1-8 have all been completed in the previous steps followed in the practical sheet.
  - a. You may be asked to enter your Github credentials when you do above.
  - b. However, entering your password wont work here. Instead you will have to use a Personal Access Token (PAT). Follow this [documentation](#) to setup the PAT. Make sure this PAT is copied and saved somewhere safe. It will only be visible once.
  - c. Then try step 9 again, and when prompted for password enter your generated PAT instead.
  - d. If all went well, your code should now be pushed and be available on GitHub.

# IT3030 (PAF) – Practical Sheet 2

## Version Controlling with Git - II

### Introduction

Last week, we went through the basic commands in Git. This week, we are going to see one of the most important and useful features in Git: branching.

Along with branching, we will also see how to use a simple workflow (branching strategy) as well. You will learn a new concept called Pull Requests (PRs) in this practical session.

Since we have set up our git clients on our computers and GitHub accounts last week, we are just going to use them in this session.

### Step 1 – Individual work

1. Create a Public GitHub Repository.
2. Then **Clone** it to your computer using the Git Clone command (`git clone <https://repo_url>`)
3. Once the cloning is complete, create a feature branch in the local repository.
  - a. There are two options for creating a branch.
    - i. `git branch <meaningful_branch_name>` - just creates the branch based on the current branch. Use `git checkout <new_branch>` to check out the new branch.
    - ii. `git checkout -b <meaningful_branch_name>` - creates the branch and immediately switches to the created branch.
    - iii. Make sure to give a meaningful branch name and follow a good naming convention. Here is an example.  
E.g.: `git checkout -b feature/vishan.j/awesome-feature-x-to-our-app`
4. Again, add some text files/ html files (content doesn't matter) to this branch.
5. Commit changes in this branch locally and then push this branch to the remote.
6. Ensure your changes are available in the remote repository on GitHub if all went well. The changes should be available under your feature branch name.
7. Use [the GitHub official documentation](#) to create a Pull Request on Github for your branch.
8. Then merge the pull request to the main by referring to [this official documentation](#).

### Step 2 – Pair work

1. Now pair up with a colleague (preferably sitting next to you) and add them to your repository as a collaborator by referring to [this documentation](#).
2. Now clone their repository to your computer.
3. Then as with step 1, create a branch and introduce a small change.
4. Push the branch to the remote repository.
5. Use [the GitHub official documentation](#) to create a Pull Request on Github for your branch.
6. Then merge the pull request created by your friend to the main by referring to [this official documentation](#).

# IT3030 (PAF) – Practical Sheet 3

## Introduction to Spring Framework

*Note: Look up the meanings of the terms in **bold**.*

### Introduction

#### What is a framework?

A framework enables a coding environment that contains low-level libraries to address conventional coding issues. The objective of a framework is to deliver faster development of an application. This includes everything we need to build large-scale applications, such as templates based on best practices.

#### What is Spring?

Spring is a comprehensive framework for building Java applications. Spring is well known for being very configurable and less **opinionated** thereby allowing the users of the framework complete control over how the framework should behave according to the users' needs.

Although this flexibility has its advantages, when a need arises to create an application quickly, the same flexibility can become a hinderance which is the problem solved by Spring Boot.

#### What is Spring Boot?

As mentioned before, too much choice can also be hinderance if you want just everything to work out of the box. That is exactly the need Spring Boot caters to. Spring Boot is an **opinionated** extension of the Spring framework with working configurations out of the box along with third-party library **integrations** allowing developers to focus on implementing actual business logic.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

### Prerequisites

Please have the following software set up in your computer before continuing.

#### Essential Software

- Java Development Kit (JDK) 18 or later
- Visual Studio Code (VS Code)

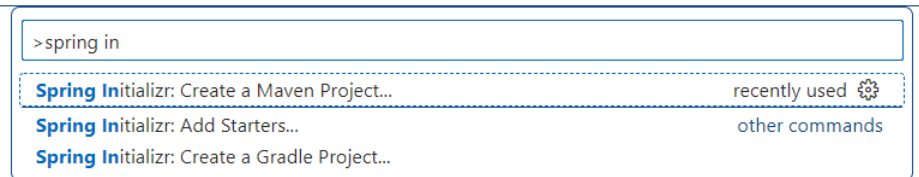
After VS Code is set up, please install the following extensions to VS Code as well.

- Spring Initializr v0.11.2 (by Microsoft)
- Extension Pack for Java v0.25.7 (by Microsoft)

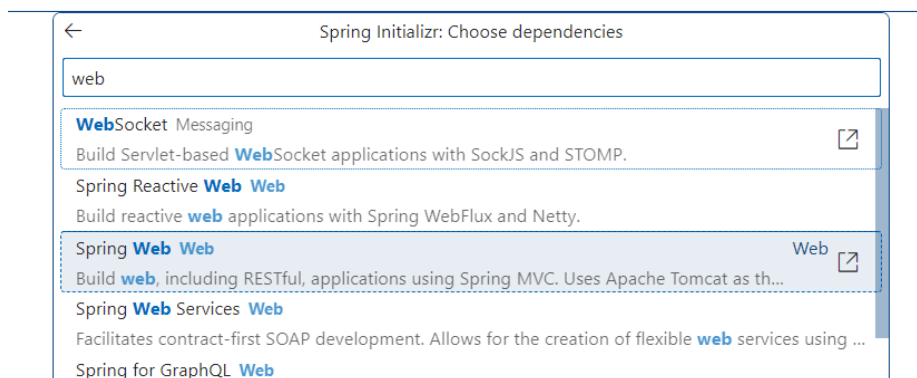
## Creating a simple Spring Boot Application

### Part 1 – Create a new Spring Boot Project

1. Open the command palette in VS Code (Ctrl + Shift + P in Windows).
2. Type “Spring initializr” and select *Spring Initializr: Create a Maven Project*



3. Select the latest Spring Boot version. Make sure not to select versions labelled as *SNAPSHOT*.
4. Select *Java* as the Project language.
5. Give a name for the *group id* and *artifact id*. See [here](#) for an explanation on these.
6. Select *Jar* as the packaging type for your project.
7. Select the *Java Version* (select one which has been installed on your computer)
8. Select *Dependencies* for the project. For this exercise, select only the *Spring Web* dependency.



9. Then press enter and select the folder path for creating the Spring Boot project.

### Part 2 – Creating a Hello World! Application

1. Open the relevant VS Code project and locate the `<artifactid>Application.java` available under `src/main/java/com/<groupid>/<artifactid>` and open it.
2. Add the method `rootEndpoint()` and the **annotations** (highlighted) in this file just after the main method (see the image below).

```
@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/")
    public String rootEndpoint() {
        String message = "Hello world!";
        return message;
    }
}
```

The `@RestController` annotation tells Spring that this code describes an **endpoint** that should be made available over the web. The `@GetMapping("/")` tells Spring to use our `rootEndpoint()` method to answer requests that get sent to the `http://localhost:8080/` address.

3. Some errors will be thrown regarding missing packages etc. Make sure those are fixed using the QuickFix option in VS Code.
4. Once all errors have been fixed, run the project on VS Code. It should execute successfully.
5. Open your browser and navigate to <http://localhost:8080>. You will see "Hello world!" displayed on the page.

Note: In case you run into an error due to port numbers, open *application.properties* file and add this line: *server.port=8081*. Save and re-run the project.

### Part 3 – Self Guided Activity

1. Modify the above file to respond to an endpoint *"/hello"*.  
E.g., *http://localhost:8080/hello*.
2. Then, modify the method to display a name passed on as a **Query Parameter** along with Hello.  
E.g., *http://localhost:8080/hello?name=Amith*

Hint: [See this link for help](#).

# IT3030 (PAF) – Practical Sheet 4

## Creating and consuming a simple web API

Note: Look up the meanings of the terms in **bold**.

### Introduction

Last week, a simple introduction to spring and spring boot was made. This week, we will create two simple programs. One to serve greetings over a web API when called and the other to consume this service.

### What is an API?

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols ([Source](#)). In this particular case, it is via http.

### Prerequisites

- To have completed the practical sheet 4 – Introduction to Spring framework.
- Optional: Install Postman API platform tool.

### Part 1: Creating the API

1. Create a new Spring Boot project using the *Spring initializr* through VS code. For this exercise, select only the *Spring Web* dependency. Name the project as *GreetingAPI*. Please refer the previous week's practical sheet for more information on creating a new project.
2. The resource with which we are going to respond to a call to our API is a Greeting. The greeting should consist of a greeting along with a unique id for each greeting. Create a new file called *Greeting.java* to represent a model greeting we are going to respond with. Inside this file, create a new class called *Greeting*. It is called a *resource representation class*.

```
J Greeting.java
1 package com.example.greetingapi;
2
3 // a record class for a greeting response.
4 // read more: https://docs.oracle.com/en/java/javase/15/language/records.html
5 public record Greeting(long id, String content) { }
6
```

*Greeting class*

3. You may notice that the structure of the class is quite different from a normal class structure in Java that is familiar to you. This type of class is called a **Record class** and they are intended to act as “data carriers”. Functionally it is still similar to a regular class in Java, but more concise. Read more at [this link](#).
4. In Spring's approach to building web APIs, HTTP requests are handled by a controller. These components are identified by the *@RestController* annotation. A Controller is simply a class which has methods and logic to handle calls made to predefined **endpoints** in our web API. Create new file called *GreetingController.java*. Implement the *GreetingController* class in this file. Make sure to include the appropriate dependencies when writing code for the class.



```

9 // controller for Greeting related endpoints
10 @RestController
11 public class GreetingController {
12
13     private static final String template = "Hello, %s!";
14
15     private final AtomicLong count = new AtomicLong();
16
17     // endpoint for responding to calls for /greeting
18     @GetMapping("/greeting")
19     public Greeting greeting() {
20         return new Greeting(count.incrementAndGet(), String.format(template, "World"));
21     }
22
23     // endpoint for responding to calls for /greeting/name?name=<your_name>
24     @GetMapping("/greeting/name")
25     public Greeting greeting(@RequestParam(value = "name", defaultValue = "<Your name>") String name) {
26         return new Greeting(count.incrementAndGet(), String.format(template, name));
27     }
28
29 }

```

*GreetingController class – observe the coding style. @RestController/ @GetMapping annotations were discussed in the previous lab sheet.*

5. Save everything and run the project.
6. Once the project is running, navigate to <http://localhost:8080/greeting> through the web browser and observe the response.
7. Then navigate to [http://localhost:8080/greeting/name?name=<your\\_name\\_here>](http://localhost:8080/greeting/name?name=<your_name_here>) and observe the response.

Note: For steps 6 and 7, the Postman tool may be used instead of the web browser.

## Part 2: Creating the API consuming application

It's nice to be able to access a resource through a browser or through a tool, but it is not very useful. A more useful way of consuming a resource is via a program which is going to be the next step.

1. Create a new Spring Boot project via the *Spring initializr* through VS code. Name the project as Getgreetings.
2. To contain the data that is to be received from the GreetingAPI, we need to have a *domain class*. Create *Greeting.java* and implement the Greeting class. This again is a *record class*.

```

J Greeting.java
1 package com.example.getgreetings;
2
3 // a record class for a greeting response.
4 // read more: https://docs.oracle.com/en/java/javase/15/language/records.html
5 public record Greeting(long id, String content) { }
6

```

*Greeting class*

3. Then, in the *GetgreetingsApplication* class, add the methods *getHttpClient()*, *getGreeting()*, *getGreetingByName()*, *makeCalls()* to consume the services in API as below. A comment above each method explains what each of them is for.

```

7  @SpringBootApplication
8  public class GetgreetingsApplication {}
9
10     // client for performing HTTP requests
11     private static RestTemplate httpClient = null;
12
13     // base url for remote calls
14     private static String baseUrl = "http://localhost:8080/";
15
16     // endpoints for remote calls
17     private static String defaultGreetingURL = "greeting";
18     private static String namedGreetingURL = "greeting/name?name=<your_name_here>";
19
20     // main method
21 > public static void main(String[] args) { ...
25     }
26
27     // singleton pattern implemented to get a single instance of the http client
28 > private static RestTemplate getHttpClient() { ...
34     }
35
36     // call the default endpoint and get the response
37 > private static Greeting getGreeting(String url) { ...
42     }
43
44     // call the named endpoint and get the response
45 > private static Greeting getGreetingByName(String url) { ...
50     }
51
52     // call the endpoints, receive the responses and print them on the console
53 > private static void makeCalls() { ...
62     }

```

*Overall structure of GetgreetingsApplication class - observe the coding style.*

4. The main method once expanded looks like this. It only contains a method called *makeCalls()*.

```

20     // main method
21     public static void main(String[] args) {
22         SpringApplication.run(GetgreetingsApplication.class, args);
23
24         makeCalls();
25     }
26

```

5. The `makeCalls()` method looks like this. It makes calls to `getGreeting()` and `getGreetingByName()` methods, then prints the contents of each response.

```
52 // call the endpoints, receive the responses and print them on the console
53 private static void makeCalls() {
54     Greeting receivedGreeting1 = GetgreetingsApplication.getGreeting(defaultGreetingURL);
55     Greeting receivedGreeting2 = GetgreetingsApplication.getGreetingByName(namedGreetingURL);
56
57     String content1 = receivedGreeting1.content();
58     System.out.println(content1);
59
60     String content2 = receivedGreeting2.content();
61     System.out.println(content2);
62 }
--
```

6. `getGreeting()` and `getGreetingByName()` methods look like this. Both of them call `getHttpClient()` which returns an object of type **RestTemplate**. Read about *RestTemplate* [here](#).

```
36 // call the default endpoint and get the response
37 private static Greeting getGreeting(String url) {
38     RestTemplate restmp = getHttpClient();
39     Greeting response = restmp.getForObject(baseUrl + "/" + url, Greeting.class);
40
41     return response;
42 }
43
44 // call the named endpoint and get the response
45 private static Greeting getGreetingByName(String url) {
46     RestTemplate restmp = getHttpClient();
47     Greeting response = restmp.getForObject(baseUrl + "/" + url, Greeting.class);
48
49     return response;
50 }
51
```

7. The `getHttpClient()` method looks like this.

```
27 // singleton pattern implemented to get a single instance of the http client
28 private static RestTemplate getHttpClient() {
29
30     if (httpClient == null) {
31         httpClient = new RestTemplate();
32     }
33     return httpClient;
34 }
```

8. Complete the class and try to run the program. Please include the dependencies as required.  
9. Does the program run? If it does not, try and identify the cause of the issue and fix it.  
10. Once the issue is fixed, try running it again.

### Part 3: Self-learning activity

1. Add a new endpoint in the API program to send today's date along with the greeting.
2. Modify the Getgreetings application to receive this new type of greeting as well.
3. Can the API you developed be considered as a REST API? Find out.

Note: The original API endpoints also should work along with the new endpoint.

# IT3030 (PAF) – Practical Sheet 5

## REST APIs with Spring

The six architectural constraints in REST were discussed in this week's lecture. This practical sheet aims to give you an introduction to creating APIs which align with the six architectural constraints in REST.

### Six Architectural constraints in REST

1. Client-server architecture
2. Stateless
3. Cacheable
4. Uniform Interface
5. Layered system
6. Code on demand (Optional)

Refer the *REST APIs* lecture slide deck for more information.

### Prerequisites

We have had a gentle introduction to Spring and Spring Boot for two weeks now.

1. You should have completed all the practical sheets up to now.
2. You should have Java, VS Code and Postman set up successfully in your computer.

### Part 1: A simple introduction to Spring HATEOAS

1. By referring this [official Spring documentation](#) create a very simple Hypermedia Driven REST API which will accept GET requests at <http://localhost:8080/greeting> and respond with a response like below, which is a valid RESTful response.

```
{
  "content": "Hello, World!",
  "_links": {
    "self": {
      "href": "http://localhost:8080/greeting?name=World"
    }
  }
}
```

2. After you are done typing the code, the code can be run straight away on VS code. You can skip the *creating executable JAR* step for this exercise.

### Self-learning activity: Building REST services with Spring

1. Follow this [official Spring documentation](#) on building proper REST services using Spring on your own and complete it. It is very descriptive with regards to what is happening in the code.
2. The tutorial is divided into four sections.
  - a. Non-REST — Simple Spring MVC app with no hypermedia.
  - b. REST — Spring MVC + Spring HATEOAS app with HAL representations of each resource.
  - c. Evolution — REST app where a field is evolved but old data is retained for backward-compatibility.
  - d. Links — REST app where conditional links are used to signal valid state changes to clients.

Make sure to complete all the sections.

3. Do not blindly copy paste all the code. Understand what each piece of code does and proceed.

### Some observations regarding the tutorial

1. You may encounter an issue with importing `javax` namespace. Replace it with `jakarta`.  
E.g. replace `javax.persistence.Entity`; with `jakarta.persistence.Entity`;
2. You may want to refer a certain `linkTo()` method in `WebMvcLinkBuilder`. In case you encounter any issue with discovering this method, include this import:

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;
```

# IT3030 (PAF) – Practical Sheet 6

## JavaScript Basics

- Objective: To teach a set of basic concepts in the JavaScript programming language.
  - Prerequisites: Students should have basic JavaScript knowledge.
1. JavaScript Objects

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<!--In JavaScript, an object is a standalone entity, with properties
and type.-->
<p id="demo"></p>

<script>

// Animal properties and method encapsulation
const Animal = {
  type: "Invertebrates", // Default value of properties
  displayType() {
    // Method which will display type of Animal
    console.log(this.type);
  },
};

// Create new animal type called animal1
const animal1 = Object.create(Animal);
animal1.displayType(); // Logs: Invertebrates

// Create new animal type called fish
const fish = Object.create(Animal);
fish.type = "Fishes";
fish.displayType(); // Logs: Fishes

</script>

</body>
```

## 2. JavaScript Closure

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Closure</h2>

<!--
  A closure is a function having access to the parent scope,
  even after the parent function has closed.
-->

<p id="demo"></p>

<script>

//a closure gives you access to an outer function's scope from an
inner function.

function greeting() {
  let message = 'Hi';

  function sayHi() {
    console.log(message);
  }

  return sayHi;
}
let hi = greeting();
hi(); // still can access the message variable'

</script>

</body>
</html>
```

### 3. JSON Placeholder API

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON Placeholder API</h2>

<!--An application programming interface is a way for two or more
computer programs to communicate with each other.
    It is a type of software interface,
    offering a service to other pieces of software. -->

<p id="demo"></p>

<script>

//https://jsonplaceholder.typicode.com/
//Free fake API for testing and prototyping.

fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))

</script>

</body>
</html>
```



## ES6 New features

1. Classes
  - a. Create a simple class constructor.

```
<!DOCTYPE html>
<html>

<body>

<script>

//What is this? In JavaScript, the this keyword refers to an
object.
//Which object depends on how this is being invoked (used or
called).
//The this keyword refers to different objects depending on how it
is used:
//In an object method, this refers to the object.

class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}

const mycar = new Car("Ford");
document.write(mycar.present());
</script>

</body>
</html>
```

- b. Create a class and define a method inside the class, after that, create an object from the class and execute the methods.

```
<!DOCTYPE html>
<html>

<body>

<script>
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}

const mycar = new Car("Ford");
document.write(mycar.present());
</script>

</body>
</html>
```

- c. Class inheritance – create a class (base class) and create another class, derived from base class that you created and make a method within each class and, execute method within derived class by creating an object of derived class and then, execute the base class's method via that object.

```
<!DOCTYPE html>
<html>
<body>
<script>
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}

class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model
  }
}

const mycar = new Model("Ford", "Mustang");
document.write(mycar.show());
</script>
</body>
</html>
```

## 2. Variables

- a. “var”, “let” and “const” variables. Try their behaviors.

```
<!DOCTYPE html>
<html>

<body>

<script>

let a = 10;
  function f() {
    if (true) {
      let b = 9

      // It prints 9
      console.log(b);
    }

    // It gives error as it
    // defined in if block
    console.log(b);
  }
  f()

  // It prints 10
  console.log(a)

</script>

<p>Press F12 and see the result in the console view.</p>

</body>
</html>
```

```
<!DOCTYPE html>
<html>

<body>

<script>

const a = {
    prop1: 10,
    prop2: 9
}

// It is allowed
a.prop1 = 3

// It is not allowed
a = {
    b: 10,
    prop2: 9
}

</script>
```

<p>Press F12 and see the result in the console view.</p>

```
</body>
</html>
```

### 3. Array methods

- a. Map a list of items from an array.

```
<!DOCTYPE html>
<html>

<body>

  <h1 id="demo"></h1>

  <script>
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map(x => x * 2);

document.getElementById("demo").innerHTML = map1;
// Expected output: Array [2, 8, 18, 32]

</script>

</body>
</html>
```

#### 4. Destructuring

- a. Use destructuring when a function returns an array.

```
<!DOCTYPE html>
<html>

<body>

<script>
function calculate(a, b) {
  const add = a + b;
  const subtract = a - b;
  const multiply = a * b;
  const divide = a / b;

  return [add, subtract, multiply, divide];
}

const [add, subtract, multiply, divide] = calculate(4, 7);

document.write("<p>Sum: " + add + "</p>");
document.write("<p>Difference " + subtract + "</p>");
document.write("<p>Product: " + multiply + "</p>");
document.write("<p>Quotient " + divide + "</p>");
</script>

</body>
</html>
```

- b. Destructure deeply nested objects by referencing the nested object then using a colon and curly braces to again destructure the items needed from the nested object.

```
<!DOCTYPE html>
<html>

<body>

<p id="demo"></p>

<script>
const vehicleOne = {
  brand: 'Ford',
  model: 'Mustang',
  type: 'car',
  year: 2021,
  color: 'red',
  registration: {
    city: 'Houston',
    state: 'Texas',
    country: 'USA'
  }
}

myVehicle(vehicleOne)

function myVehicle({ model, registration: { state } }) {
  const message = 'My ' + model + ' is registered in ' + state +
  '.';

  document.getElementById("demo").innerHTML = message;
}
</script>

</body>
</html>
```



## Self-study activity - Asynchronous JavaScript

Self-study these concepts and attempt the following exercises.

1. JavaScript Callbacks - Write a simple callback function for following scenario

“The fetchData function takes in a URL and a callback function as parameters. It makes a request to the specified URL using the XMLHttpRequest API, and then calls the callback function with the response data (or an error) when the request is complete. The fetchData function is used with a callback function that logs the response data to the console if there are no errors or logs the error to the console if there is one.”

2. JavaScript Promises - Write a simple promise for following scenario

“The fetchData function returns a Promise object that wraps an XMLHttpRequest. The Promise object is either resolved with the response data or rejected with an error message, depending on the result of the request. The fetchData function is used with a then method that logs the response data to the console if the Promise is resolved, or a catch method that logs the error to the console if the Promise is rejected.”

3. JavaScript async & await - Write the code that explains following scenario

“The fetchDataAsync function is an asynchronous function that uses the await keyword to wait for the Promise returned by fetchData to be resolved or rejected. The try...catch block is used to handle any errors that occur during the execution of the function. The fetchDataAsync function is used to fetch data from a URL and log it to the console if there are no errors, or log the error to the console if there is one.”