Unit V

Multithreading in Java

UNIT V

Multithreading in Java

Syllabus

Concurrency and Synchronization, Java Thread Model: Thread priorities, Synchronization, Messaging, Main Thread, Creating thread: Implementing Thread using thread class and Runnable interface. Creating multiple threads using is Alive() and join(). **Web Based Application in Java:** Use of JavaScript for creating web based applications in Java, Introduction to Java script frameworks- ReactJS, VueJS, AngularJS (open source).

JAVA THREAD

Thread Definition

- **A Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.
- A thread is a subpart of a process that can run individually**.**

In order to perform complicated tasks in the background, we used the Thread concept in Java. All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent. Another benefit of using **thread** is that if a thread gets an exception or an error at the time of its execution, it doesn't affect the execution of the other threads. All the threads share a common memory and have their own stack, local variables and program counter. When multiple threads are executed in parallel at the same time, this process is known as **Multithreading.**

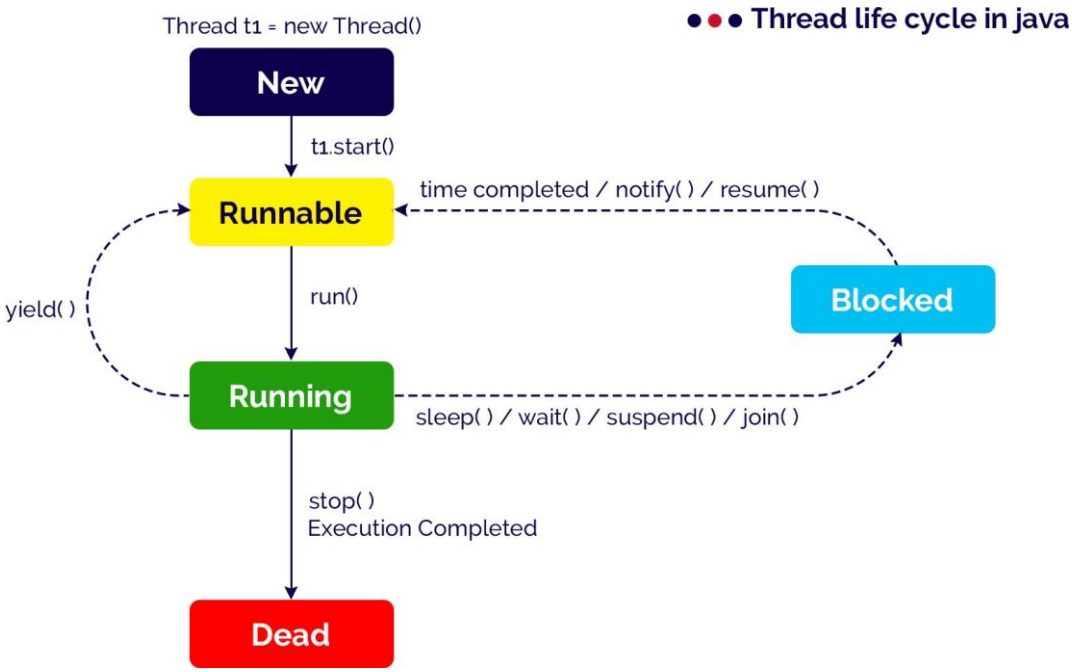
In a simple way, a Thread is a:

- Feature through which we can perform multiple activities within a single process.
- Lightweight process.
- Series of executed statements.
- Nested sequence of method calls.

Thread Model

In java, a thread goes through different states throughout its execution. These stages are called thread life cycle states or phases. A thread may in any of the states like new, ready or

runnable, running, blocked or wait, and dead or terminated state. The life cycle of a thread in java is shown in the following figure.



1) New

A thread is in **New** when it gets CPU time. When a thread object is created using new, then the thread is said to be in the New state. This state is also known as Born state.

Example

```
Thread t1 = new Thread();
```

2) Runnable / Ready

A thread is in **a Runnable** state when it is under execution. When a thread calls start( ) method, then the thread is said to be in the Runnable state. This state is also known as a Ready state.

Example

```
t1.start();
```

3) Running

When a thread calls **run( )** method, then the thread is said to be **Running**. The run( ) method of a thread called automatically by the start( ) method.

4) Suspended

A thread is in the **Suspended** state when it is temporarily inactive or under execution.

5) Blocked

A thread is in the **Blocked** state when it is waiting for resources. A thread in the Running state may move into the blocked state due to various reasons like sleep( ) method called, wait( ) method called, suspend( ) method called, and join( ) method called, etc.
When a thread is in the blocked or waiting state, it may move to Runnable state due to reasons like sleep time completed, waiting time completed, notify( ) or notifyAll( ) method called, resume( ) method called, etc.

Example

```
Thread.sleep(1000);
wait(1000);
wait();
suspened();
notify();
notifyAll();
resume();
```

6) Dead / Terminated

A thread comes in this state when at any given time, it halts its execution immediately. A thread in the Running state may move into the dead state due to either its execution completed or the **stop( )** method called. The dead state is also known as the **terminated** state.

Creating Thread

In java, a thread is a lightweight process. Every java program executes by a thread called the main thread. When a java program gets executed, the main thread created automatically. All other threads called from the main thread.
The java programming language provides two methods to create threads, and they are listed below.

- **Using Thread class (by extending Thread class)**
- **Uisng Runnable interface (by implementing Runnable interface)**

1) Extending Thread class

The java contains a built-in class Thread inside the **java.lang** package. The Thread class contains all the methods that are related to the threads.
To create a thread using Thread class, follow the step given below.

- **Step-1**: Create a class as a child of Thread class. That means, create a class that extends Thread class.
- **Step-2**: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.
- **Step-3**: Create the object of the newly created class in the main( ) method.
- **Step-4**: Call the start( ) method on the object created in the above step.

Look at the following example program.

```
class SampleThread extends Thread{

	public void run() {
		System.out.println("Thread is under Running...");
		for(int i= 1; i<=10; i++) {
			System.out.println("i = " + i);
		}
	}
}

public class My_Thread_Test {

	public static void main(String[] args) {
		SampleThread t1 = new SampleThread();
		System.out.println("Thread about to start...");
		t1.start();
	}
}
```

Output

```
Thread about to start...
Thread is under Running...
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

2) Implementng Runnable interface

The java contains a built-in interface Runnable inside the java.lang package. The Runnable interface implemented by the Thread class that contains all the methods that are related to the threads.

To create a thread using Runnable interface, follow the step given below.

- **Step-1**: Create a class that implements Runnable interface.

- **Step-2**: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.
- **Step-3**: Create the object of the newly created class in the main( ) method.
- **Step-4**: Create the Thread class object by passing above created object as parameter to the Thread class constructor.
- **Step-5**: Call the start( ) method on the Thread class object created in the above step.

Look at the following example program.

```
class SampleThread implements Runnable{

	public void run() {
		System.out.println("Thread is under Running...");
		for(int i= 1; i<=10; i++) {
			System.out.println("i = " + i);
		}
	}
}

public class My_Thread_Test {

	public static void main(String[] args) {
		SampleThread threadObject = new SampleThread();
		Thread thread = new Thread(threadObject);
		System.out.println("Thread about to start...");
		thread.start();
	}
}
```

Output

```
Thread about to start...
Thread is under Running...
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

More about Thread class

The Thread class in java is a subclass of Object class and it implements Runnable interface. The Thread class is available inside the java.lang package. The Thread class has the following syntax.

```
class Thread extends Object implements Runnable{
...
}
```

The Thread class has the following consructors.

- **Thread( )**
- **Thread(String threadName)**
- **Thread(Runnable objectName)**
- **Thread(Runnable objectName, String threadName)**

The Thread classs contains the following methods.

Method	**Description**	**Return Value**
run( )	Defines actual task of the thread.	void
start( )	It moves thre thread from Ready state to Running state by calling run( ) method.	void
setName(String)	Assigns a name to the thread.	void
getName( )	Returns the name of the thread.	String
setPriority(int)	Assigns priority to the thread.	void
getPriority( )	Returns the priority of the thread.	int
getId( )	Returns the ID of the thread.	long
activeCount()	Returns total number of thread under active.	int
currentThread( )	Returns the reference of the thread that currently in running state.	void
sleep(long)	moves the thread to blocked state till the specified number of milliseconds.	void
isAlive( )	Tests if the thread is alive.	boolean
yield( )	Tells to the scheduler that the current thread is willing to yield its current use of a processor.	void
join( )	Waits for the thread to end.	void

Thread Priorities

In a java programming language, every thread has a property called priority. Most of the scheduling algorithms use the thread priority to schedule the execution sequence. In java, the thread priority range from 1 to 10. Priority 1 is considered as the lowest priority, and priority 10 is considered as the highest priority. The thread with more priority allocates the processor first.

The java programming language Thread class provides two methods **setPriority(int)**, and **getPriority( )** to handle thread priorities.

The Thread class also contains three constants that are used to set the thread priority, and they are listed below.

- ***MAX_PRIORITY*** - It has the value 10 and indicates highest priority.
- ***NORM_PRIORITY*** - It has the value 5 and indicates normal priority.
- ***MIN_PRIORITY*** - It has the value 1 and indicates lowest priority.

The default priority of any thread is 5 (i.e. NORM_PRIORITY).

setPriority( ) method

The setPriority( ) method of Thread class used to set the priority of a thread. It takes an integer range from 1 to 10 as an argument and returns nothing (void).

The regular use of the setPriority( ) method is as follows.

Example

```
threadObject.setPriority(4);
```

or

```
threadObject.setPriority(MAX_PRIORITY);
```

getPriority( ) method

The getPriority( ) method of Thread class used to access the priority of a thread. It does not takes anyargument and returns name of the thread as String. The regular use of the getPriority( ) method is as follows.

Example

```
String threadName = threadObject.getPriority();
```

Look at the following example program.

```
class SampleThread extends Thread{
       public void run() {
              System.out.println("Inside SampleThread");
              System.out.println("Current Thread: " +
Thread.currentThread().getName());
       }
}
```

```
public class My_Thread_Test {

	public static void main(String[] args) {
		SampleThread threadObject1 = new SampleThread();
		SampleThread threadObject2 = new SampleThread();
		threadObject1.setName("first");
		threadObject2.setName("second");

		threadObject1.setPriority(4);
		threadObject2.setPriority(Thread.MAX_PRIORITY);

		threadObject1.start();
		threadObject2.start();

	}
}
```

Output

```
Inside SampleThread
Current Thread: second
Inside SampleThread
Current Thread: first
```

In java, it is not guaranteed that threads execute according to their priority because it depends on JVM specification that which scheduling it chooses.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (**Multiprocessing**)
- Thread-based Multitasking (**Multithreading**)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

It is important to know the difference between process-based and thread-based multitasking. Let's distinguish both.

Process-based multitasking (Multiprocessing)	Thread-based multitasking (Multithreading)
It allows the computer to run two or more programs concurrently	It allows the computer to run two or more threads concurrently
In this process is the smallest unit.	In this thread is the smallest unit.
Process is a larger unit.	Thread is a part of process.
Process is heavy weight.	Thread is light weight.
Process requires seperate address space for each.	Threads share same address space.
Process never gain access over idle time of CPU.	Thread gain access over idle time of CPU.
Inter process communication is expensive.	Inter thread communication is not expensive.

Synchronization

Multithreading in Java is a process of executing multiple threads simultaneously. The problem of shared resources occurs when two or more threads get execute at the same time. In such a situation, we need some way to ensure that the shared resource will be accessed by only one thread at a time, and this is performed by using the concept called **synchronization**.

The synchronization is the process of allowing only one thread to access a shared resource at a time.

In java, the synchronization is achieved using the following concepts.

- Synchronized Blocks
- Synchronized methods

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

1. **Using Synchronized Blocks**

Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks. You keep shared resources within this block. Following is the general form of the synchronized statement –

Syntax

```
synchronized(objectidentifier) {
   // Access shared variables and other shared resources
}
```

Here, the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents. Now we are going to see two examples, where we will print a counter using two different threads. When threads are not synchronized, they print counter value which is not in sequence, but when we print counter by putting inside synchronized() block, then it prints counter very much in sequence for both the threads.

Example

Without Synchronization

```
class PrintDemo {
   public void printCount() {
      try {
        for(int i = 3; i > 0; i--) {
System.out.println("C --- "  + i );
         }
      } catch (Exception e) {
 System.out.println("interrupted.");
      }
   }
}
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;
ThreadDemo(String name,  PrintDemo
pd) {
      threadName = name;
      PD = pd;
   }
public void run() {
      PD.printCount();
System.out.println("Thread " +
threadName + " exiting.");
   }


   public void start () {
System.out.println("Starting " +
threadName );
 if (t == null) {
  t = new Thread (this, threadName);
        t.start ();
      }
```

With Synchronization

```
class PrintDemo {
   public void printCount() {
      try {
         for(int i = 5; i > 0; i--) {
System.out.println("C --- "  + i );
         }
      } catch (Exception e) {
System.out.println("interrupted.");
      }
   }
}
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;
ThreadDemo( String name,  PrintDemo
pd) {
      threadName = name;
      PD = pd;
   }
public void run() {
      synchronized(PD) {
         PD.printCount();
      }
System.out.println("Thread " +
threadName + " exiting.");
   }
   public void start () {
      System.out.println("Starting "
+  threadName );
      if (t == null) {
   t = new Thread (this, threadName);
    t.start ();
      }
```

`}`
`}`
`public class TestThread {`
`  public static void main(String args[]) {`
`  PrintDemo PD = new PrintDemo();`
`  ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );`
`  ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );`
`  T1.start();`
`  T2.start();`
`  // wait for threads to end`
`  try {`
`  T1.join();`
`  T2.join();`
`  } catch ( Exception e) {`
`  System.out.println("Interrupted");`
`  }`
`  }`
`}`
	`}`
	`}`
	`public class TestThread {`
	`  public static void main(String args[]) {`
	`  PrintDemo PD = new PrintDemo();`
	`  ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );`
	`  ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );`
	`  T1.start();`
	`  T2.start();`
	`  // wait for threads to end`
	`  try {`
	`  T1.join();`
	`  T2.join();`
	`  } catch ( Exception e) {`
	`  System.out.println("Interrupted");`
	`  }`
	`  }`
	`}`

Output
`Starting Thread - 1`
`Starting Thread - 2`
`Counter --- 3`
`Counter --- 2`
`Counter --- 1`
`Counter --- 3`
`Counter --- 2`
`Counter --- 1`
`Thread Thread - 2 exiting.`
`Thread Thread - 1 exiting.`
	Output
	`Starting Thread - 1`
	`Starting Thread - 2`
	`Counter --- 3`
	`Counter --- 2`
	`Counter --- 1`
	`Thread Thread - 1 exiting.`
	`Counter --- 3`
	`Counter --- 2`
	`Counter --- 1`
	`Thread Thread - 2 exiting.`

This produces a different result every time you run this program	This produces the same result every time you run this program

2. Using Synchronized methods

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Without Synchronization	With Synchronization

`class PrintDemo {`
`  public void printCount() {`
`  try {`
`  for(int i = 3; i > 0; i--) {`
`System.out.println("C --- " + i );`
`  }`
`  } catch (Exception e) {`
	`class PrintDemo {`
	`  public synchronized void printCount() {`
	`  try {`
	`  for(int i = 5; i > 0; i--) {`
	`System.out.println("C --- " + i );`
	` }

```
 System.out.println("interrupted.");
      }
   }
}
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;
ThreadDemo(String name,  PrintDemo
pd) {
      threadName = name;
      PD = pd;
   }
public void run() {
      PD.printCount();
System.out.println("Thread " +
threadName + " exiting.");
   }


   public void start () {
System.out.println("Starting " +
threadName );
 if (t == null) {
  t = new Thread (this, threadName);
         t.start ();
      }
   }
}
public class TestThread {
   public static void main(String
args[]) {
    PrintDemo PD = new PrintDemo();
    ThreadDemo T1 = new ThreadDemo(
"Thread - 1 ", PD );
    ThreadDemo T2 = new ThreadDemo(
"Thread - 2 ", PD );
      T1.start();
      T2.start();
      // wait for threads to end
         try {
         T1.join();
         T2.join();
      } catch ( Exception e) {
  System.out.println("Interrupted");
      }
   }
}
```

Output

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   3
Counter   ---   2
Counter   ---   1
Counter   ---   3
Counter   ---   2
```

```
      } catch (Exception e) {
System.out.println("interrupted.");
      }
   }
}
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;
ThreadDemo( String name,  PrintDemo
pd) {
      threadName = name;
      PD = pd;
   }
public void run() {
        PD.printCount();
System.out.println("Thread " +
threadName + " exiting.");
   }
   public void start () {
      System.out.println("Starting "
+  threadName );
      if (t == null) {
   t = new Thread (this, threadName);
    t.start ();
      }
   }
}
public class TestThread {
   public static void main(String
args[]) {
      PrintDemo PD = new PrintDemo();
      ThreadDemo T1 = new ThreadDemo(
"Thread - 1 ", PD );
      ThreadDemo T2 = new ThreadDemo(
"Thread - 2 ", PD );
      T1.start();
      T2.start();
      // wait for threads to end
      try {
         T1.join();
         T2.join();
      } catch ( Exception e) {
  System.out.println("Interrupted");
      }
   }
}
```

Output

```
Starting Thread - 1
Starting Thread - 2
Counter   ---   3
Counter   ---   2
Counter   ---   1
Thread Thread - 1  exiting.
Counter   ---   3
```

`Counter --- 1`
`Thread Thread - 2 exiting.`
`Thread Thread - 1 exiting.`
	`Counter --- 2`
	`Counter --- 1`
	`Thread Thread - 2 exiting.`

This produces a different result every time you run this program	This produces the same result every time you run this program

isAlive() And join() Methods in Java Multi-Threading

When we are using multiple threads in our application, we'll create the threads and call the *start()* method on these threads. It's the Java Virtual Machine that calls the run method of this thread when the resources and CPU are ready. Then the thread will execute the *run()* method and later transition to terminated state.

If you want to start further processing only when a thread has ended, in that scenario how will you know that a thread has ended. Java multi-threading provides two ways to find that–

- isAlive()
- join()

1. isAlive() method in Java

isAlive() method is the member of the Thread class and its general form is–

```
public final boolean isAlive()
```

isAlive() method tests if the thread it is called upon is alive or not. A thread is alive if it has been started and not yet terminated. The isAlive() method returns true if the thread upon which it is called is still running, otherwise it returns false.

2. join() method in Java

join() method is used when you want to wait for the thread to finish. Its general form is–

```
public final void join() throws InterruptedException
```

This method waits until the thread on which it is called terminates. There are three overloaded join functions.

- `public final void join()` – Waits indefinitely for this thread to die.
- `public final void join(long millis)` – Waits at most millis milliseconds for this thread to die. A timeout of 0 means to wait forever.
- `public final void join(long millis, int nanos)` – Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.

Example code using join and isAlive()

```
class MyRunnableClass implements Runnable{
    @Override
    public void run() {
        for(int i = 0; i < 3 ; i++){
            System.out.println(Thread.currentThread().getName() + " i - "
+ i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
public class JoinExample {
    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnableClass(), "t1");
        Thread t2 = new Thread(new MyRunnableClass(), "t2");
        Thread t3 = new Thread(new MyRunnableClass(), "t3");

        t1.start();
        t2.start();
        t3.start();

        System.out.println("t1 Alive - " + t1.isAlive());
        System.out.println("t2 Alive - " + t2.isAlive());
        System.out.println("t3 Alive - " + t3.isAlive());

        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        System.out.println("t1 Alive - " + t1.isAlive());
        System.out.println("t2 Alive - " + t2.isAlive());
        System.out.println("t3 Alive - " + t3.isAlive());

        System.out.println("Processing finished");
    }
}
```

Output

```
t1 Alive - true
t2 Alive - true
t3 Alive - true
t3 i - 0
t2 i - 0
t1 i - 0
```

```
t3 i - 1
t2 i - 1
t1 i - 1
t3 i - 2
t1 i - 2
t2 i - 2
t1 Alive - false
t2 Alive - false
t3 Alive - false
Processing finished
```

Now see how message is displayed only when the processing is actually finished and all the threads are terminated. The second print statements using isAlive() method confirms that the threads are not running any more.

WEB BASED APPLICATION IN JAVA

JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is

a structured programming language.

3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

1. Client-side validation,
2. Dynamic drop-down menus,
3. Displaying date and time,
4. Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
5. Displaying clocks etc.

Hello World using Javascript

Code	Output
<html> <body> <script language = "javascript" type = "text/javascript"> <!-- document.write("Hello World!") //--> </script> </body> </html>	Hello World!

There are many useful Javascript frameworks and libraries available:

- Angular
- React
- jQuery
- Vue.js
- Ext.js
- Ember.js
- Meteor
- Mithril
- Node.js
- Polymer
- Aurelia
- Backbone.js

Client-Side JavaScript

- Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
- The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are –

- Less server interaction – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors – They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features –

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.
- Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

Syntax

JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

You can place the <script> tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
```

```
   JavaScript code
</script>

<script language = "javascript" type = "text/javascript">
   JavaScript code
</script>
```

Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type = "text/javascript">
   <!--
      var money;
      var name;
   //-->
</script>
```

You can also declare multiple variables with the same var keyword as follows –

```
<script type = "text/javascript">
   <!--
      var money, name;
   //-->
</script>
```

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

JavaScript Reserved Words

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	FALSE	native	throws
catch	final	new	transient
char	finally	null	TRUE
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with

Data Types

To be able to operate on variables, it is important to know something about the type. Without data types, a computer cannot safely solve this:

```
let x = 16 + “Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + “Volvo";
```

Example

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

```
<h2>JavaScript Data Types</h2>

<p>JavaScript has dynamic types. This means that the same variable can be
used to hold different data types:</p>

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

<script>
let x;          // Now x is undefined
x = 5;          // Now x is a Number
x = "Selva";     // Now x is a String

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

</body>
</html>
```

Output

JavaScript Data Types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Selva

Statements

JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments. This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Code

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

```
<h2>JavaScript Statements</h2>

<p>JavaScript statements are separated by semicolons.</p>

<p id="demo1"></p>

<script>
let a, b, c;
a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>

</body>
</html>
```

Output

JavaScript Statements

JavaScript statements are separated by semicolons.

11

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
}
```

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

Calculate the product of two numbers, and return the result:

```
let x = myFunction(4, 3);   // Function is called, return value will
end up in x
function myFunction(a, b) {
  return a * b;             // Function returns the product of a and
b
}
```

The result in x will be:

12

Objects

- In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property.
- The key of a property can be a string. And the value of a property can be any value, e.g., a string, a number, an array, and even a function.
- JavaScript provides you with many ways to create an object. The most commonly used one is to use the object literal notation.

The following example creates an empty object using the object literal notation:

```
let empty = {};
```

To create an object with properties, you use the ***key:value*** within the curly braces. For example, the following creates a new person object:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

The person object has two properties firstName and lastName with the corresponding values 'John' and 'Doe'.

When an object has multiple properties, you use a comma (,) to separate them like the above example.

Accessing properties

To access a property of an object, you use one of two notations: the dot notation and array-like notation.

The dot notation (.)

The following illustrates how to use the dot notation to access a property of an object:

```
objectName.propertyName
```

For example, to access the firstName property of the person object, you use the following expression:

```
person.firstName
```

This example creates a person object and shows the first name and last name to the console:

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
console.log(person.firstName);
console.log(person.lastName);
```

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

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
```

```
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Output

JavaScript Objects

John is 50 years old.

Embedding JavaScript in HTML

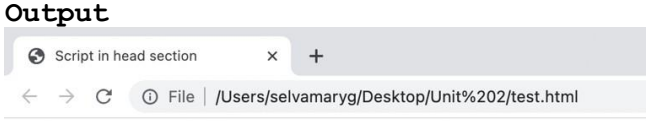
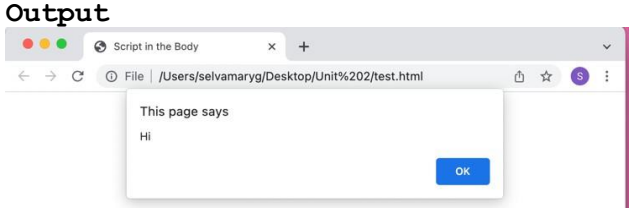
There are two general areas in HTML document where JavaScript can be placed.

- First is between <head>……..</head> section,
- another is specific location in <body>……. </body> section.
- Using the script tag to include an external JavaScript file

If you want to display a message 'Good Morning' (through the JavaScript alert command) at the time of page loading then you must place the script at the <head>. …… </head> section. In the following examples you will see the different location of <script>.....</script> tags in a HTML document.

Script in the Head	**Script in the Body**
<!DOCTYPE html> <head> <meta charset="utf-8" /> <title> Script in head section</title> <script type = "text/javascript"> JavaScript statements……. </script> </head>	<!DOCTYPE html> <head> <meta charset="utf-8" /> <title> Script in the Body</title> </head> <body> <script type = "text/javascript"> Alert(“hi”)

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

Output 	**Output** 

Using the script tag to include an external JavaScript file

To include an external JavaScript file, we can use the script tag with the attribute src. You've already used the src attribute when using images. The value for the src attribute should be the path to your JavaScript file.

```
<script type="text/javascript" src="path-to-javascript-file.js"></script>
```

This script tag should be included between the <head> tags in your HTML document.

JavaScript Files

JavaScript files are not HTML files or CSS files.

- Always end with the js extension
- Only include JavaScript

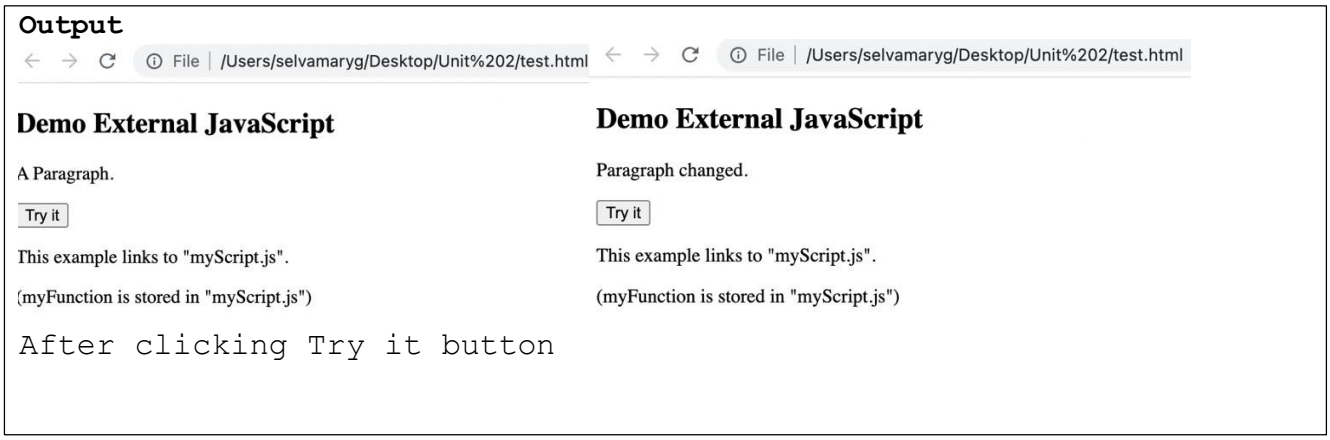
External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External file: test.html

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

<h2>Demo External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

Output



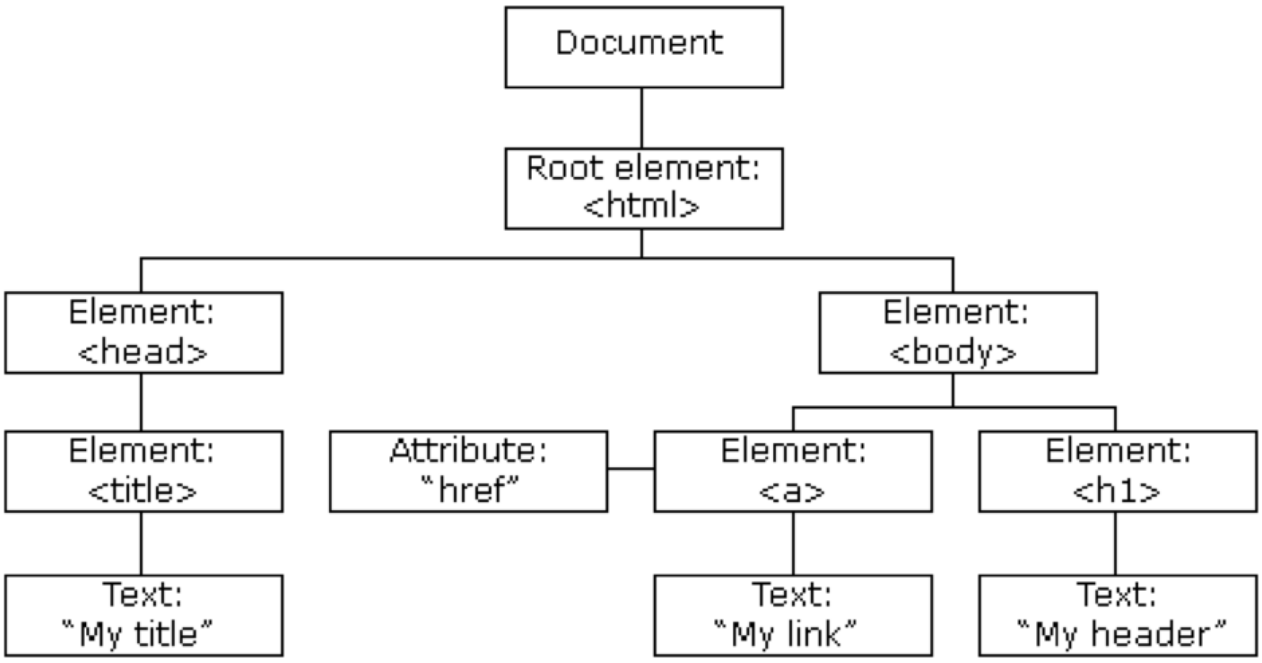
After clicking Try it button

Document Object Model

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

DOM TREE :

The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

ReactJS

Currently, ReactJS is one of the most popular JavaScript front-end libraries which has a strong foundation and a large community.

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

Our ReactJS tutorial includes all the topics which help to learn ReactJS. These are ReactJS Introduction, ReactJS Features, ReactJS Installation, Pros and Cons of ReactJS, ReactJS JSX, ReactJS Components, ReactJS State, ReactJS Props, ReactJS Forms, ReactJS Events, ReactJS Animation and many more.

The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the

apps. It uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses component and data patterns that improve readability and helps to maintain larger apps.

React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem. The important features of ReactJS are as following

1. **JSX:** JSX is a JavaScript syntax extension. The JSX syntax is processed into JavaScript calls of React Framework. It extends the ES6 so that HTML like text can co-exist with JavaScript React code.
2. **Components:** ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its logic and controls. These components can be reusable, which help you to maintain the code when working on larger scale projects.
3. **One-way Data Binding:** ReactJS follows unidirectional data flow or one-way data binding. The one-way data binding gives you better control throughout the application. If the data flow is in another direction, then it requires additional features. It is because components are supposed to be immutable, and the data within them cannot be changed.
4. **Virtual DOM:** A virtual DOM object is a representation of the real DOM object. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then, it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that are changed. It makes the application faster, and there is no wastage of memory.
5. **Simplicity:** ReactJS uses the JSX file, which makes the application simple and to code as well as understand. Also, ReactJS is a component-based approach which makes the code reusable as your need. It makes it simple to use and learn.
6. **Performance:** ReactJS is known to be a great performer. The reason behind this is that it manages a virtual DOM. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM. Instead, we are writing virtual Components that will turn into the DOM, leading to smoother and faster performance.

Pros and Cons of ReactJS

Today, ReactJS is the highly used open-source JavaScript Library. It helps in creating impressive web apps that require minimal effort and coding. The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. There are important pros and cons of ReactJS given as following:

Advantage of ReactJS

1. Easy to Learn and USe

ReactJS is much easier to learn and use. It comes with a good supply of documentation, tutorials, and training resources. Any developer who comes from a JavaScript background can easily understand and start creating web apps using React in a few days. It is the V(view part) in the MVC (Model-View-Controller) model, and referred to as ?one of the JavaScript frameworks.? It is not fully featured but has the advantage of open-source JavaScript User Interface(UI) library, which helps to execute the task in a better manner.

2. Creating Dynamic Web Applications Becomes Easier

To create a dynamic web application specifically with HTML strings was tricky because it requires a complex coding, but React JS solved that issue and makes it easier. It provides less coding and gives more functionality. It makes use of the JSX(JavaScript Extension), which is a particular syntax letting HTML quotes and HTML tag syntax to render particular subcomponents. It also supports the building of machine-readable codes.

3. Reusable Components

A ReactJS web application is made up of multiple components, and each component has its own logic and controls. These components are responsible for outputting a small, reusable piece of HTML code which can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

4. Performance Enhancement

ReactJS improves performance due to virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. Most of the developers faced the problem when the DOM was updated, which slowed down the performance of the application. ReactJS solved this problem by introducing virtual DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM. Instead, we are writing virtual components that react will turn into the DOM, leading to smoother and faster

performance.

5. The Support of Handy Tools

React JS has also gained popularity due to the presence of a handy set of tools. These tools make the task of the developers understandable and easier. The React Developer Tools have been designed as Chrome and Firefox dev extension and allow you to inspect the React component hierarchies in the virtual DOM. It also allows you to select particular components and examine and edit their current props and state.

6. Known to be SEO Friendly

Traditional JavaScript frameworks have an issue in dealing with Search Engine Optimization (SEO). The search engines generally having trouble in reading JavaScript-heavy applications. Many web developers have often complained about this problem. ReactJS overcomes this problem that helps developers to be easily navigated on various search engines. It is because React.js applications can run on the server, and the virtual DOM will be rendering and returning to the browser as a regular web page.

7. The Benefit of Having JavaScript Library

Today, ReactJS is choosing by most of the web developers. It is because it is offering a very rich JavaScript library. The JavaScript library provides more flexibility to the web developers to choose the way they want.

8. Scope for Testing the Codes

ReactJS applications are extremely easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

Disadvantage of ReactJS

1. The high pace of development

The high pace of development has an advantage and disadvantage both. In case of disadvantage, since the environment continually changes so fast, some of the developers not feeling comfortable to relearn the new ways of doing things regularly. It may be hard for them to adopt all these changes with all the continuous updates. They need to be always updated with their skills and learn new ways of doing things.

2. Poor Documentation

It is another cons which are common for constantly updating technologies. React technologies updating and accelerating so fast that there is no time to make proper documentation. To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.

3. View Part

ReactJS Covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

4. JSX as a barrier

ReactJS uses JSX. It's a syntax extension that allows HTML with JavaScript mixed together. This approach has its own benefits, but some members of the development community consider JSX as a barrier, especially for new developers. Developers complain about its complexity in the learning curve.

React Environment Setup

In this section, we will learn how to set up an environment for the successful development of ReactJS application.

Pre-requisite for ReactJS

1. NodeJS and NPM
2. React and React DOM
3. Webpack
4. Babel

Ways to install ReactJS

There are two ways to set up an environment for successful ReactJS application. They are given below.

1. Using the npm command
2. Using the create-react-app command

Using create-react-app

If you have npx and Node.js installed, you can create a React application by using create-react-app.

If you've previously installed create-react-app globally, it is recommended that you uninstall the package to ensure npx always uses the latest version of create-react-app.

To uninstall, run this command:

```
npm uninstall -g create-react-app.
```

Run this command to create a React application named my-react-app:

```
npx create-react-app my-react-app
```

The create-react-app will set up everything you need to run a React application.

Run the React Application

Now you are ready to run your first *real* React application!

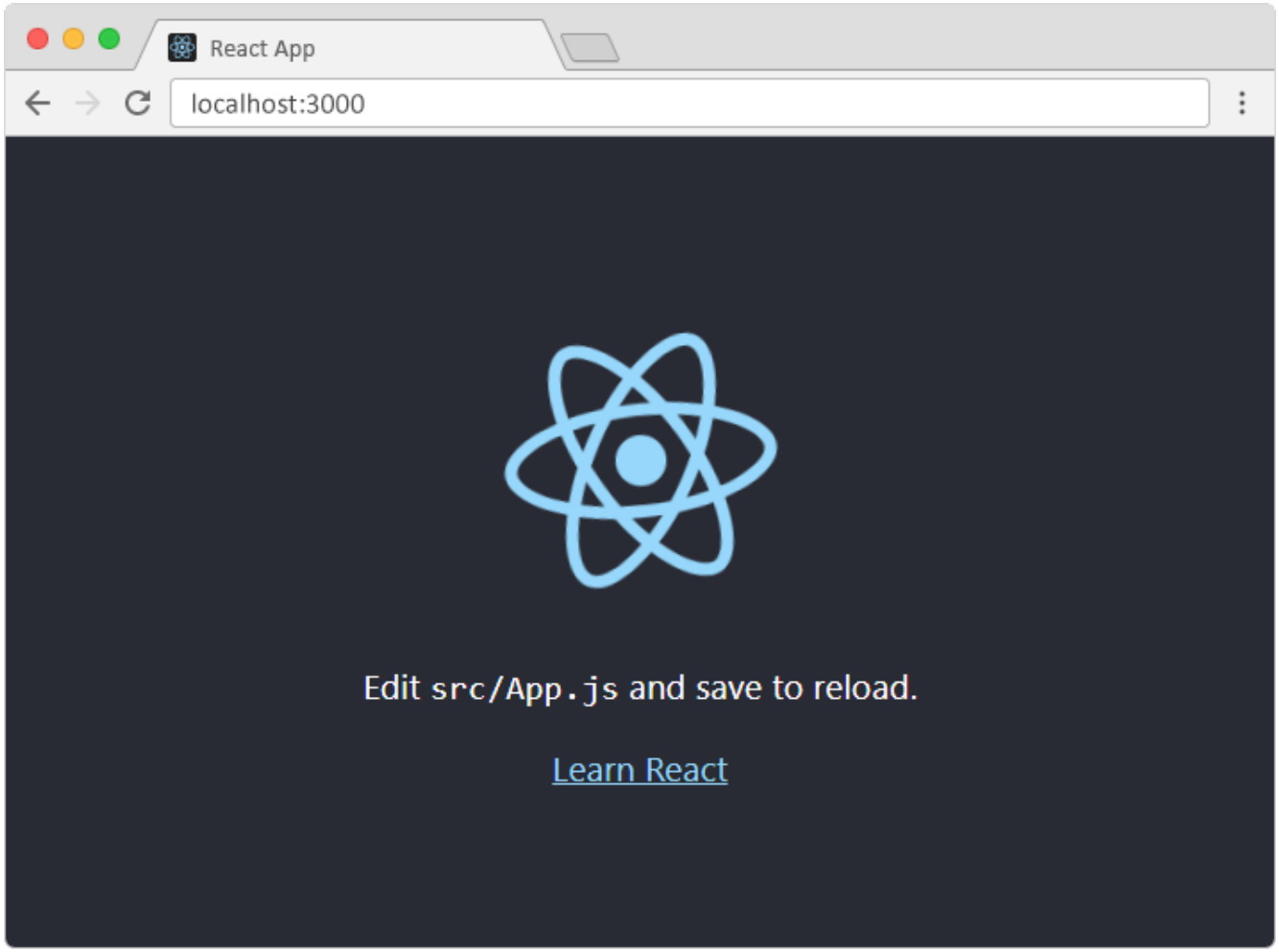
Run this command to move to the my-react-app directory:

```
cd my-react-app
```

Run this command to run the React application my-react-app:

```
npm start
```

A new browser window will pop up with your newly created React App! If not, open your browser and type **localhost:3000** in the address bar.



Modify the React Application

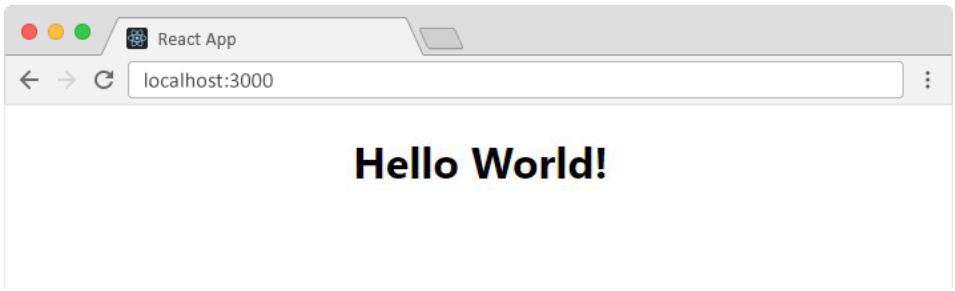
So far so good, but how do I change the content?

Look in the my-react-app directory, and you will find a src folder. Inside the src folder there is a file called App.js, open it and it will look like this:

/myReactApp/src/App.js:

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

Output



Angular JS

AngularJS is a very powerful JavaScript Framework. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions. AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Features of Angular JS

1. **Data-binding:** AngularJS follows the two-way data binding. It is the automatic synchronization of data between model and view components.
2. **POJO Model:** AngularJS uses POJO (Plain Old JavaScript) model, which provides spontaneous and well-planned objects. The POJO model makes AngularJS self-sufficient and easy to use.
3. **Model View Controller(MVC) Framework:** MVC is a software design pattern used for developing web applications. The working model of AngularJS is based on MVC patterns. The MVC Architecture in AngularJS is easy, versatile, and dynamic. MVC makes it easier to build a separate client-side application.
4. **Services:** AngularJS has several built-in services such as $http to make an XMLHttpRequest.

5. **User interface with HTML:** In AngularJS, User interfaces are built on HTML. It is a declarative language which has shorter tags and easy to comprehend. It provides an organized, smooth, and structured interface.
6. **Dependency Injection:** AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.
7. **Active community on Google:** AngularJS provides excellent community support. It is Because Google maintains AngularJS. So, if you have any maintenance issues, there are many forums available where you can get your queries solved.
8. **Routing:** Routing is the transition from one view to another view. Routing is the key aspect of single page applications where everything comes in a single page. Here, developers do not want to redirect the users to a new page every time they click the menu. The developers want the content load on the same page with the URL changing.

Why to Learn AngularJS?

- AngularJS is an open-source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.2.21.
- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.
- Overall, AngularJS is a framework to build large scale, high-performance, and easyto-maintain web applications.

Hello World using AngularJS.

```
<html>
   <head>
      <title>AngularJS First Application</title>
   </head>
   <body>
      <h1>Sample Application</h1
```

```
      <div ng-app = "">
         <p>Enter your Name: <input type = "text" ng-model = "name"></p>
         <p>Hello <span ng-bind = "name"></span>!</p>
      </div>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
      </script>
   </body>
</html>
```

Output

Angular 7 Environment Setup

- In this page, you will see how you can install the prerequisites needed to run your first Angular 7 app.
- Install Visual Studio Code IDE or JetBrains WebStorm
- You must have an IDE like Visual Studio Code IDE or JetBrains WebStorm to run your Angular 7 app.
- VS Code is light and easy to setup, it has a great range of built-in code editing, formatting, and refactoring features. It is free to use. It also provides a huge number of extensions that will significantly increase your productivity.
- You can download VS Code from here: https://code.visualstudio.com

Install Node.js

You should install node.js to run your Angular 7 app. It manages npm dependencies support some browsers when loading particular pages. It provides required libraries to run Angular project. Node.js serves your run-time environment as your localhost.

See how to install node.js: install-nodejs

Or

Just go to node.js official website https://nodejs.org/en/

After the successful installation, you will see command prompt like this:

Use npm to install Angular CLI

Run the Angular CLI command to install Angular CLI

```
npm install -g @angular/cli
```

or

Just go to Angular CLI official website https://cli.angular.io/

You will see the whole cli command to create an Angular app. You need to run the first command to install Angular CLI. These steps are same for Windows and Mac.

```
1. npm install -g @angular/cli
2. ng new my-dream-app
3. cd my-dream-app
4. ng serve
```

Your Angular 7 Environment setup is complete now.

AngularJS Vs. ReactJS

	AngularJS	**ReactJS**
Author	Google	Facebook Community
Developer	Misko Hevery	Jordan Walke
Initial Release	October 2010	March 2013
Latest Version	Angular 1.7.8 on 11 March 2019.	React 16.8.6 on 27 March 2019
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open Source JS Framework
Rendering	Client-Side	Server-Side
Packaging	Weak	Strong
Data-Binding	Bi-directional	Uni-directional
DOM	Regular DOM	Virtual DOM
Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux
Dependencies	It manages dependencies automatically.	It requires additional tools to manage dependencies.

Routing	It requires a template or controller to its router configuration, which has to be managed manually.	It doesn't handle routing but has a lot of modules for routing, eg., react-router.
Performance	Slow	Fast, due to virtual DOM.
Best For	It is best for single page applications that update a single view at a time.	It is best for single page applications that update multiple views at a time.

VueJS

- Vue.js lets you **extend** HTML with HTML attributes called **directives**
- Vue.js directives offers **functionality** to HTML applications
- Vue.js provides **built-in** directives and **user defined** directives

Vue.js Directives

Vue.js uses double braces {{ }} as place-holders for data.

Vue.js directives are HTML attributes with the prefix v-

VueJS is an open source progressive JavaScript framework used to develop interactive web interfaces. It is one of the famous frameworks used to simplify web development. VueJS focusses on the view layer. It can be easily integrated into big projects for front-end development without any issues.

The installation for VueJS is very easy to start with. Any developer can easily understand and build interactive web interfaces in a matter of time. VueJS is created by Evan You, an ex- employee from Google. The first version of VueJS was released in Feb 2014. It recently has clocked to 64,828 stars on GitHub, making it very popular.

Features

Following are the features available with VueJS.

1. **Virtual DOM**

VueJS makes the use of virtual DOM, which is also used by other frameworks such as React, Ember, etc. The changes are not made to the DOM, instead a replica of the DOM is created which is present in the form of JavaScript data structures. Whenever any changes are to be made, they are made to the JavaScript data structures and the latter is compared with the original data structure. The final changes are then updated to the real DOM, which the user will see changing. This is good in terms of optimization, it is less expensive and the changes

can be made at a faster rate.

2. **Data Binding**

The data binding feature helps manipulate or assign values to HTML attributes, change the style, assign classes with the help of binding directive called **v-bind** available with VueJS.

3. **Components**

Components are one of the important features of VueJS that helps create custom elements, which can be reused in HTML.

4. **Event Handling**

v-on is the attribute added to the DOM elements to listen to the events in VueJS.

5. **Animation/Transition**

VueJS provides various ways to apply transition to HTML elements when they are added/updated or removed from the DOM. VueJS has a built-in transition component that needs to be wrapped around the element for transition effect. We can easily add third party animation libraries and also add more interactivity to the interface.

6. **Computed Properties**

This is one of the important features of VueJS. It helps to listen to the changes made to the UI elements and performs the necessary calculations. There is no need of additional coding for this.

7. **Templates**

VueJS provides HTML-based templates that bind the DOM with the Vue instance data. Vue compiles the templates into virtual DOM Render functions. We can make use of the template of the render functions and to do so we have to replace the template with the render function.

8. **Directives**

VueJS has built-in directives such as v-if, v-else, v-show, v-on, v-bind, and v-model, which are used to perform various actions on the frontend.

9. **Watchers**

Watchers are applied to data that changes. For example, form input elements. Here, we don't have to add any additional events. Watcher takes care of handling any data changes making the code simple and fast.

10. **Routing**

Navigation between pages is performed with the help of vue-router.

11. **Lightweight**

VueJS script is very lightweight and the performance is also very fast.

12. **Vue-CLI**

VueJS can be installed at the command line using the vue-cli command line interface. It helps to build and compile the project easily using vue-cli.

Vue Example

In the example below, a new Vue object is created with **new Vue()**.

The property **el:** binds the new Vue object to the HTML element with **id="app"**.

Example

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>

<div id="app">
  <h1>{{ message }}</h1>
</div>

<script>
var myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue!'}
})
</script>
</body>
</html>
```

Output

Hello Vue!

Example

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>
<h2>Vue.js</h2>
<div id="app">
  {{ message }}
</div>

<p>
<button onclick="myFunction()">Click Me to greet you!</button>
</p>

<script>
var myObject = new Vue({
  el: '#app',
```

```
  data: {message: 'Hello Vue!'}
})

function myFunction() {
    myObject.message = "Hello Selva";
}
</script>

</body>
</html>
```

Output

After clicking the button

****