Web Technologies — Full Code Guide (All 24 Practicals)

Each practical has a full working code example, run tips, and quick notes. Open the contained code in appropriate environments (browser, Node.js, or React app) and follow the run instructions on each page.

Practical 1 — Static Website (Hyperlinks, Formatting, Images, Multimedia, Lists)

Run tip: Save as `index.html` and open in browser.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Static Site</title>
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <style>
    body{font-family:Arial, sans-serif;line-height:1.6;padding:20px}
    header{border-bottom:2px solid #333;padding-bottom:10px;margin-bottom:10px}
    img{max-width:100%;height:auto;border-radius:6px}
    nav ul{list-style:circle;margin-left:20px}
    .card{border:1px solid #ddd;padding:12px;border-radius:6px;margin:8px 0}
  </style>
</head>
<body>
  <header>
    <h1>My Static Website</h1>
    <p>A demo with links, images, multimedia, and lists.</p>
  </header>

  <nav>
    <a href="#about">About</a> |
    <a href="#media">Media</a> |
    <a href="#links">Useful Links</a>
  </nav>

  <section id="about" class="card">
    <h2>About</h2>
    <p><strong>Formatting:</strong> Use &lt;strong&gt;, &lt;em&gt;, headings and paragraphs.</p>
    <ul>
      <li>Item 1</li>
      <li>Item 2 with <em>emphasis</em></li>
      <li>Nested list:
        <ol>
          <li>Step A</li>
          <li>Step B</li>
        </ol>
      </li>
    </ul>
    <img src="https://via.placeholder.com/600x200.png?text=Sample+Image" alt="sample" />
  </section>

  <section id="media" class="card">
    <h2>Multimedia</h2>
    <p>Video and audio from public sources (for demo only):</p>
    <video width="560" controls>
      <source src="https://www.w3schools.com/html/mov_bbb.mp4" type="video/mp4">
      Your browser does not support the video tag.
    </video>
    <br/>
    <audio controls>
      <source src="https://www.w3schools.com/html/horse.mp3" type="audio/mpeg">
      Your browser does not support the audio element.
    </audio>
  </section>

  <section id="links" class="card">
    <h2>Useful Links</h2>
    <ul>
      <li><a href="https://developer.mozilla.org/" target="_blank" rel="noopener">MDN Web Docs</a></li>
      <li><a href="#about">Jump to About</a></li>
    </ul>
  </section>
</body>
</html>
```

Practical 2 — Webpages with CSS3 (Backgrounds, Fonts, Tables, Lists, Selectors)

Run tip: Save `css3.html` and `styles.css` in same folder and open css3.html

css3.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>CSS3 Demo</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header class="hero">
    <h1>CSS3 Features</h1>
  </header>

  <section>
    <h2>Table</h2>
    <table class="styled-table">
      <thead><tr><th>Name</th><th>Role</th></tr></thead>
      <tbody>
        <tr><td>Jay</td><td>Student</td></tr>
        <tr><td>Aisha</td><td>Developer</td></tr>
      </tbody>
    </table>
  </section>

  <section>
    <h2>Lists & Selectors</h2>
    <ul class="fancy-list">
      <li>Alpha</li>
      <li>Beta</li>
      <li>Gamma</li>
    </ul>
  </section>
</body>
</html>
```

styles.css

```
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

body{font-family:'Roboto',sans-serif;margin:0;padding:20px;background:linear-gradient(180deg,#f0f5ff,#fff);color:#2
.hero{background:url('https://via.placeholder.com/1200x200.png?text=Background') center/cover no-repeat;padding:40p
.styled-table{width:100%;border-collapse:collapse;margin-top:10px}
.styled-table th, .styled-table td{padding:8px;border:1px solid #ddd;text-align:left}
.styled-table thead th{background:#27406b;color:#fff}
.fancy-list{list-style-image:url('https://via.placeholder.com/16/0000FF/FFFFFF?text=•');margin-left:20px}
```

Practical 3 — CSS Border & Color Properties

Run tip: Save as `border_color_demo.html` and open in browser.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Border & Color</title>
  <style>
    .box{width:300px;height:100px;padding:10px;margin:10px;font-family:Arial}
    .rounded{border:4px solid #ff4500;border-radius:12px;background:#fff3ee}
    .shadow{border:2px dashed #0066cc;background:linear-gradient(#e6f3ff,#fff);box-shadow:0 4px 8px rgba(0,0,0,0.1)
  </style>
</head>
<body>
  <div class="box rounded">Rounded border, orange</div>
  <div class="box shadow">Dashed border with shadow</div>
</body>
</html>
```

Practical 4 — JavaScript Registration Form Validation (email + required)

Run tip: Save as `register.html` and open in browser.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Registration Validation</title>
  <style>input{display:block;margin:8px 0;padding:8px;width:300px}</style>
</head>
<body>
  <h2>Register</h2>
  <form id="regForm" novalidate>
    <input id="name" name="name" placeholder="Full name" required>
    <input id="email" name="email" placeholder="Email" required>
    <input id="password" name="password" type="password" placeholder="Password" required>
    <button type="submit">Submit</button>
  </form>
  <div id="msg" role="status"></div>

  <script>
    const form = document.getElementById('regForm');
    const msg = document.getElementById('msg');

    function validEmail(email){
      return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
    }

    form.addEventListener('submit', e => {
      e.preventDefault();
      const name = form.name.value.trim();
      const email = form.email.value.trim();
      const pw = form.password.value.trim();

      if(!name || !email || !pw){
        msg.textContent = 'All fields are required.';
        msg.style.color='red'; return;
      }
      if(!validEmail(email)){
        msg.textContent = 'Invalid email address.';
        msg.style.color='red'; return;
      }
      msg.textContent = 'Registration successful!';
      msg.style.color='green';
      // submit to server via fetch/ajax if needed
    });
  </script>
</body>
</html>
```

Practical 5 — JavaScript Iterators & Generators

Run tip: Save as `iterators_generators.js` and run with `node iterators_generators.js`

```
// Iterator example (manual)
const arr = [10,20,30];
const it = arr[Symbol.iterator]();
console.log(it.next()); // {value:10,done:false}
console.log(it.next()); // {value:20,done:false}
console.log(it.next()); // {value:30,done:false}

// Generator example
function* idGenerator(){
  let id = 1;
  while(true){
    yield id++;
  }
}
const gen = idGenerator();
console.log(gen.next().value); //1
console.log(gen.next().value); //2
console.log(gen.next().value); //3

// Use a generator to produce limited sequence
function* take(n){
  for(let i=0;i<n;i++) yield i*i;
}
for(const x of take(5)) console.log('square:', x);
```

Practical 6 — JavaScript Promise Demo (then/catch and async/await)

Run tip: Save as `promise_demo.js` and run with Node.

```
function wait(ms){
  return new Promise((resolve) => setTimeout(()=> resolve(`Waited ${ms}ms`), ms));
}

// using then/catch
wait(500).then(res => console.log('then:', res)).catch(err => console.error(err));

// using async/await
(async function(){
  try {
    const r1 = await wait(300);
    console.log('await:', r1);
  } catch(e) {
    console.error(e);
  }
})();

// Example of Promise.all and Promise.race
Promise.all([wait(100), wait(200)]).then(vals => console.log('all:', vals));
Promise.race([wait(150), wait(300)]).then(val => console.log('race winner:', val));
```

Practical 7 — JavaScript Inheritance (ES6 classes)

Run tip: Save as `inheritance.js` and run with Node.

```
class Person {
  constructor(name){
    this.name = name;
  }
  greet(){
    return `Hello, I'm ${this.name}`;
  }
}

class Student extends Person {
  constructor(name, roll){
    super(name);
    this.roll = roll;
  }
  details(){
    return `${this.greet()} (Roll: ${this.roll})`;
  }
}

const s = new Student('Jay', 30);
console.log(s.details());

// Prototype-based example
function Animal(type){
  this.type = type;
}
Animal.prototype.speak = function(){ return 'I am a ' + this.type; }

function Dog(name){ Animal.call(this,'Dog'); this.name = name; }
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Dog.prototype.bark = function(){ return this.name + ' says woof'; }

const d = new Dog('Rex');
console.log(d.speak(), d.bark());
```

Practical 8 — React: Class and Function Components

Run tip: Use Create React App: `npx create-react-app myapp` and place these components in src/ then import into App.js

AppClass.js

```
// AppClass.js
import React, { Component } from 'react';

export default class AppClass extends Component {
  render(){
    return (
      <div>
        <h2>Class Component</h2>
        <p>This is a class component.</p>
      </div>
    );
  }
}
```

AppFunction.js

```
// AppFunction.js
import React from 'react';

export default function AppFunction(){
  return (
    <div>
      <h2>Functional Component</h2>
      <p>This is a function component.</p>
    </div>
  );
}
```

Practical 9 — React: State and Props

Run tip: Add component to React app and run `npm start`.

```
// StatePropsExample.js
import React, { useState } from 'react';

function Child({count, onIncrement}){
  return (
    <div>
      <p>Count in parent: {count}</p>
      <button onClick={onIncrement}>Increment</button>
    </div>
  );
}

export default function StatePropsExample(){
  const [count, setCount] = useState(0);
  return (
    <div>
      <h3>State & Props</h3>
      <Child count={count} onIncrement={() => setCount(c => c+1)} />
    </div>
  );
}
```

Practical 10 — React: Form Handling (Controlled Components)

Run tip: Put in React app and test with npm start.

```
// ReactForm.js
import React, { useState } from 'react';

export default function ReactForm(){
  const [form, setForm] = useState({name:'',email:''});
  const handleChange = e => setForm({...form,[e.target.name]:e.target.value});
  const handleSubmit = e => { e.preventDefault(); alert(JSON.stringify(form)); };

  return (
    <form onSubmit={handleSubmit}>
      <input name="name" value={form.name} onChange={handleChange} placeholder="Name" />
      <input name="email" value={form.email} onChange={handleChange} placeholder="Email" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Practical 11 — Fibonacci Series (REPL/Node)

Run tip: Save as `fibonacci.js` and run `node fibonacci.js`

```
// fibonacci.js
function fib(n){
  let a=0,b=1;
  const res=[];
  for(let i=0;i<n;i++){
    res.push(b);
    [a,b] = [b, a+b];
  }
  return res;
}
console.log('First 10 Fibonacci numbers:', fib(10).join(' '));

// Recursive version
function fibRec(n){
  if(n<=1) return n;
  return fibRec(n-1)+fibRec(n-2);
}
console.log('fibRec(7)=', fibRec(7));
```

Practical 12 — REPL Environment: variables & multiline expressions

Run tip: Save as `repl_demo.js` and run `node repl_demo.js` to enter a local REPL with helpers.

```
// repl_demo.js
const repl = require('repl');

const server = repl.start({prompt: 'MyREPL> '});
server.context.answer = 42;
server.context.add = (a,b) => a+b;
server.context.asyncWait = ms => new Promise(res => setTimeout(res, ms));

// Now in the REPL you can type: answer OR add(2,3) OR await asyncWait(1000)
```

Practical 13 — Node.js Read Stream & Write Stream

Run tip: Create `input.txt` with content and run `node stream_rw.js`

```
// stream_rw.js
const fs = require('fs');
const readStream = fs.createReadStream('input.txt', {encoding:'utf8'});
const writeStream = fs.createWriteStream('output.txt', {encoding:'utf8'});

readStream.on('data', chunk => {
  console.log('Got chunk:', chunk.slice(0,80));
  writeStream.write(chunk.toUpperCase());
});

readStream.on('end', () => {
  console.log('Read finished');
  writeStream.end();
});

readStream.on('error', err => console.error('Read error', err));
writeStream.on('finish', () => console.log('Write finished'));
```

Practical 14 — Node.js Web Module (HTTP server)

Run tip: Save as `server.js` and run `node server.js`, then open http://localhost:3000

```
// server.js
const http = require('http');
const server = http.createServer((req,res) => {
  res.writeHead(200, {'Content-Type':'text/html'});
  if(req.url === '/') res.end('<h1>Home</h1><p>Node http server</p>');
  else if(req.url === '/api') res.end(JSON.stringify({message:'API Response'}));
  else { res.writeHead(404); res.end('Not Found'); }
});
server.listen(3000, ()=> console.log('Server on http://localhost:3000'));
```

Practical 15 — Express Router

Run tip: `npm init -y && npm i express`, save as `express_app.js`, then run `node express_app.js`

```
// express_app.js
const express = require('express');
const app = express();
const router = express.Router();

// Simple JSON endpoint
router.get('/users', (req,res) => res.json([{id:1,name:'Jay'}]));
router.get('/users/:id', (req,res) => res.json({id:req.params.id, name:'User'+req.params.id}));

app.use('/api', router);

// Example middleware
app.use((req,res,next) => {
  console.log(req.method, req.url);
  next();
});

app.listen(3001,()=>console.log('Express running on http://localhost:3001'));
```

Practical 16 — Class Timetable & Even/Odd Checker

Run tip: Save as `timetable_evenodd.html` and open in browser.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Timetable & Even/Odd</title>
  <style>
    table{width:100%;border-collapse:collapse}
    th,td{padding:8px;border:1px solid #ddd}
    .even{background:#e6ffed}
    .odd{background:#fff1f0}
  </style>
</head>
<body>
  <h2>Class Timetable</h2>
  <table>
    <thead><tr><th>Day</th><th>9-10</th><th>10-11</th></tr></thead>
    <tbody>
      <tr><td>Mon</td><td>Maths</td><td>CS</td></tr>
      <tr><td>Tue</td><td>Physics</td><td>JS</td></tr>
    </tbody>
  </table>

  <h2>Even / Odd Checker</h2>
  <input id="num" type="number" placeholder="Enter a number">
  <button id="check">Check</button>
  <div id="out"></div>

  <script>
    document.getElementById('check').addEventListener('click', () => {
      const n = Number(document.getElementById('num').value);
      const out = document.getElementById('out');
      if(Number.isInteger(n)) out.textContent = (n % 2 === 0) ? `${n} is Even` : `${n} is Odd`;
      else out.textContent = 'Enter a valid integer';
    });
  </script>
</body>
</html>
```

Practical 17 — Pattern Printing in REPL/Node

Run tip: Save as `pattern.js` and run `node pattern.js` or paste into REPL.

```
// pattern.js
const n = 5;
for(let i=1;i<=n;i++){
  console.log('*'.repeat(i));
}
```

Practical 18 — Node.js File System (read/write/delete)

Run tip: Save as `fs_demo.js` and run `node fs_demo.js`

```
// fs_demo.js
const fs = require('fs');
const fname = 'demo.txt';

// write
fs.writeFileSync(fname, 'Hello FS Demo\n', 'utf8');
console.log('Wrote file');

// append
fs.appendFileSync(fname, 'Appended line\n', 'utf8');
console.log('Appended');

// read
const data = fs.readFileSync(fname, 'utf8');
console.log('Read content:\n', data);

// delete (uncomment to delete file)
// fs.unlinkSync(fname);
// console.log('Deleted file');
```

Practical 19 — React Router (v6) Basic Example

Run tip: `npm i react-router-dom@6` in your CRA app. Use this as App.js or import component.

```
// AppRouter.js
import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

function Home(){ return <h2>Home</h2> }
function About(){ return <h2>About</h2> }
function Contact(){ return <h2>Contact</h2> }

export default function AppRouter(){
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link> | <Link to="/contact">Contact</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/contact" element={<Contact/>}/>
      </Routes>
    </BrowserRouter>
  );
}
```

Practical 20 — Node.js Piping Streams (file copy)

Run tip: Create `bigsource.txt` then run `node pipe_copy.js`

```
// pipe_copy.js
const fs = require('fs');
const r = fs.createReadStream('bigsource.txt');
const w = fs.createWriteStream('bigdest.txt');
r.pipe(w);
w.on('finish', ()=> console.log('Piping finished - file copied'));
r.on('error', e => console.error('Read error', e));
w.on('error', e => console.error('Write error', e));
```

Practical 21 — Static Website with Tables, Lists, Forms, and Alert Messages

Run tip: Save as `site_with_form.html` and open in browser.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Site with Form</title>
  <style>input{margin:6px;padding:6px}</style>
</head>
<body>
  <h1>Event Registration</h1>
  <table border="1">
    <tr><th>Event</th><th>Date</th></tr>
    <tr><td>Workshop</td><td>10 Oct</td></tr>
  </table>

  <ul><li>Rules</li><li>Time</li></ul>

  <form id="f">
    <input name="name" placeholder="Name" required>
    <input name="email" placeholder="Email" required>
    <button type="submit">Register</button>
  </form>

  <script>
    document.getElementById('f').addEventListener('submit', e=>{
      e.preventDefault();
      alert('Registration successful!');
    });
  </script>
</body>
</html>
```

Practical 22 — Apply CSS using React (Inline, External, Internal)

Run tip: Create external.css in src/ and import it. Use component in CRA app.

```
// CssInReactExample.js
import React from 'react';
import './external.css'; // external

export default function CssInReactExample(){
  const inlineStyle = {padding:'10px', border:'2px solid #333'}; // inline

  return (
    <div>
      <h3 className="external-title">External CSS Title</h3>
      <div style={inlineStyle}>This uses inline styles</div>
      <style>{`.internal {margin-top:10px;color:#555}`}</style> {/* internal */}
      <div className="internal">This uses internal styles</div>
    </div>
  );
}

// external.css
// .external-title{color:#0b6; font-weight:700}
```

Practical 23 — Factorial of a Number (REPL/Node)

Run tip: Save as `factorial.js` and run `node factorial.js`

```
// factorial.js
function factorial(n){
  if(n<=1) return 1;
  let res=1;
  for(let i=2;i<=n;i++) res*=i;
  return res;
}
console.log('6! =', factorial(6)); // 720

// recursive
function factRec(n){ return n<=1?1:n*factRec(n-1); }
console.log('5! =', factRec(5));
```

Practical 24 — React Hooks (useState & useEffect) Demo

Run tip: Add to CRA app and run `npm start`. This demonstrates state, effect, and simulated fetch.

```
// HooksDemo.js
import React, { useState, useEffect } from 'react';

export default function HooksDemo(){
  const [count, setCount] = useState(0);
  const [data, setData] = useState(null);

  useEffect(() => {
    // simulate fetch
    let mounted = true;
    setTimeout(() => { if(mounted) setData({msg: 'Hello from server'}); }, 500);
    return () => { mounted = false; }
  }, []);

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

  return (
    <div>
      <h2>useState & useEffect Demo</h2>
      <p>Count: {count}</p>
      <button onClick={() => setCount(c => c+1)}>Increment</button>
      <div>Fetched: {data ? data.msg : 'Loading...'}</div>
    </div>
  );
}
```

Quick Notes & Run Tips (Summary)

- HTML/CSS/JS: Save files (.html/.css/.js) in a folder and open the HTML file in a browser. Use a local static serv
- Node.js: Install Node (v14+). Run scripts: `node filename.js`. For REPL usage, `node` to enter interactive prompt
- Express: Initialize project `npm init -y` then `npm i express`. Run server with `node express_app.js`.
- React (Create React App): `npx create-react-app myapp`, copy components to src/, run `npm start`. Use `npm i reac
- Files referenced by Node examples (input.txt, bigsource.txt) must exist in same folder.
- For testing fetch/async calls without network, use simulated timeouts or JSON files served by a local server.
- Common commands: `npm init -y`, `npm i express`, `npx create-react-app myapp`, `npm start`, `node <file>.js`.