[JavaScript For Automation] (CheatSheet)

1. DOM Manipulation and Web Page Interaction

- **Select an Element by ID**: `document.getElementById('id')`
- **Select Elements by Class Name**: `document.getElementsByClassName('class')`
- **Select Elements by CSS Selector**: `document.querySelectorAll('selector')`
- **Create a New Element**: `document.createElement('tagName')`
- **Remove an Element**: `element.remove()`
- **Set Element Text**: `element.textContent = 'text'`
- **Set Element HTML**: `element.innerHTML = '<div>new content</div>'`
- **Change Element Style**: `element.style.color = 'blue'`
- **Toggle a Class**: `element.classList.toggle('className')`
- **Append Child to Element**: `parent.appendChild(child)`

2. Event Handling

- **Add Click Event to Element**: `element.onclick = function() {}`
- **Add Event Listener**: `element.addEventListener('click', function() {})`
- **Remove Event Listener**: `element.removeEventListener('click', function() {})`
- **Trigger an Event Programmatically**: `element.dispatchEvent(new Event('click'))`
- **Prevent Default Action**: `event.preventDefault()`
- **Stop Event Propagation**: `event.stopPropagation()`

3. Network Requests and Fetch API

- **Make a GET Request**: `fetch('url')`
- **Make a POST Request**: `fetch('url', {method: 'POST', body: JSON.stringify(data)})`
- **Send Request with Headers**: `fetch('url', {headers: {'Content-Type': 'application/json'}})`
- **Parse JSON Response**: `fetch('url').then(response => response.json())`
- **Handle Network Errors**: `fetch('url').catch(error => console.error('Error:', error))`

4. Asynchronous Operations and Promises

- **Create a New Promise**: `new Promise((resolve, reject) => {})`

- **Resolve a Promise**: `promise.then(value => {})`
- **Reject a Promise**: `promise.catch(error => {})`
- **Await an Async Function**: `(async () => { await asyncFunction(); })()`
- **Use Promise.all for Multiple Promises**: `Promise.all([promise1, promise2])`

5. Timers and Delays

- **Set a Timeout**: `setTimeout(() => {}, 1000)`
- **Clear a Timeout**: `clearTimeout(timeoutId)`
- **Set an Interval**: `setInterval(() => {}, 1000)`
- **Clear an Interval**: `clearInterval(intervalId)`

6. Storage and Cookies

- **Set Local Storage Item**: `localStorage.setItem('key', 'value')`
- **Get Local Storage Item**: `localStorage.getItem('key')`
- **Remove Local Storage Item**: `localStorage.removeItem('key')`
- **Set a Cookie**: `document.cookie = 'name=value; expires=Fri, 31 Dec 2021 23:59:59 GMT'`
- **Read a Cookie**: `document.cookie`

7. Working with Arrays and Objects

- **Map an Array**: `array.map(item => item * 2)`
- **Filter an Array**: `array.filter(item => item > 10)`
- **Reduce an Array**: `array.reduce((total, item) => total + item, 0)`
- **Find in an Array**: `array.find(item => item === 'needle')`
- **Sort an Array**: `array.sort((a, b) => a - b)`

8. Working with Strings

- **Concatenate Strings**: `` `Hello ${name}` ``
- **Match Regular Expression**: `string.match(/regex/)`
- **Replace String Content**: `string.replace('old', 'new')`
- **Convert to Upper/Lower Case**: `string.toUpperCase(), string.toLowerCase()`
- **Trim String**: `string.trim()`

9. Working with Dates and Times

- **Get Current Date and Time**: `new Date()`

- **Format a Date**: `date.toISOString()`
- **Get Specific Date Part**: `date.getFullYear()`, `date.getMonth()`, `date.getDate()`
- **Set Date and Time**: `date.setFullYear(2021)`, `date.setHours(0)`

10. Error Handling and Debugging

- **Try-Catch Block**: `try { riskyOperation(); } catch(error) { handle error }`
- **Throwing Custom Error**: `throw new Error('Custom Error')`
- **Console Logging**: `console.log('message')`
- **Console Error Logging**: `console.error('error')`
- **Using Debugger**: `debugger`

11. Web APIs and Interfaces

- **Accessing User Location**: `navigator.geolocation.getCurrentPosition(position => {})`
- **Using WebSockets**: `new WebSocket('ws://example.com')`
- **Accessing Local Files**: `inputElement.addEventListener('change', (event) => {})`
- **Using the Clipboard API**: `navigator.clipboard.writeText('Text to copy')`
- **Accessing Camera and Microphone**: `navigator.mediaDevices.getUserMedia({ video: true, audio: true })`

12. Working with JSON

- **Stringify JSON**: `JSON.stringify(object)`
- **Parse JSON String**: `JSON.parse(string)`
- **Handling JSON in Fetch**: `fetch('url').then(response => response.json())`

13. Creating and Controlling Windows

- **Open a New Window**: `window.open('https://www.example.com')`
- **Close Current Window**: `window.close()`
- **Resize Window**: `window.resizeTo(600, 400)`

14. Browser History Manipulation

- **Navigate Back**: `history.back()`
- **Navigate Forward**: `history.forward()`

- **Adding History Entry**: `history.pushState({}, '', 'newPage.html')`

15. Form and Input Handling

- **Prevent Form Submission**: `form.onsubmit = (event) => { event.preventDefault(); }`
- **Get Form Values**: `document.forms['formName']['inputName'].value`
- **Set Input Value**: `document.getElementById('inputId').value = 'newValue'`
- **Disable a Button**: `document.getElementById('buttonId').disabled = true`

16. Manipulating CSS and Styles

- **Add a Class to an Element**: `element.classList.add('new-class')`
- **Remove a Class from an Element**: `element.classList.remove('old-class')`
- **Toggle a Class on an Element**: `element.classList.toggle('toggle-class')`
- **Change Style Property**: `element.style.backgroundColor = 'red'`

17. Interacting with Documents and Windows

- **Reload Page**: `location.reload()`
- **Redirect to Another URL**: `location.href = 'https://www.example.com'`
- **Print the Page**: `window.print()`
- **Get URL Parameters**: `new URLSearchParams(window.location.search)`

18. Animation and Visual Effects

- **Basic Animation with setInterval**: `setInterval(() => { /* animation code */ }, 100)`
- **Cancel Animation**: `clearInterval(animationId)`
- **Animate Using requestAnimationFrame**: `requestAnimationFrame(animateFunction)`

19. Security and Performance

- **Encoding URI Components**: `encodeURIComponent('parameter')`
- **Decoding URI Components**: `decodeURIComponent('parameter')`
- **Sanitizing Input**: `input.replace(/<script>.*?<\/script>/g, '')`

20. Handling Files and Blob

- **Read File as Text**: `const reader = new FileReader(); reader.readAsText(file)`
- **Create Blob from Text**: `new Blob(['text'], { type: 'text/plain' })`
- **Download Blob as File**: `const url = URL.createObjectURL(blob); anchor.href = url; anchor.download = 'filename'`

21. Interacting with Other Scripts and Pages

- **Importing Scripts**: `import('./module.js').then(module => {})`
- **PostMessage to Other Windows**: `otherWindow.postMessage('Hello', '*')`
- **Listen to Message from Other Windows**: `window.addEventListener('message', event => {})`

22. Advanced Data Structures and Algorithms

- **Implementing a Queue**: `let queue = []; queue.push(1); queue.shift();`
- **Implementing a Stack**: `let stack = []; stack.push(1); stack.pop();`
- **Using Maps for Key-Value Pairs**: `let map = new Map(); map.set('key', 'value')`
- **Using Sets for Unique Items**: `let set = new Set(); set.add('item')`

23. Web Scraping and Data Extraction

- **Extracting Data from Document**: `document.querySelectorAll('.class').forEach(el => { console.log(el.textContent) })`
- **Creating a Document from String**: `new DOMParser().parseFromString(htmlString, 'text/html')`
- **Automating Form Submission**: `document.forms[0].submit()`

24. Communication with Server and APIs

- **Sending Data to Server**: `fetch('server.php', {method: 'POST', body: formData})`
- **Polling Server for Updates**: `setInterval(() => { fetch('server.php').then(r => r.text()).then(updatePage) }, 5000)`

25. Custom Events and Observers

- **Creating a Custom Event**: `let event = new CustomEvent('my-event', { detail: { key: 'value' }})`

- **Dispatching an Event**: `element.dispatchEvent(event)`
- **Observing for Mutations**: `const observer = new MutationObserver(callback); observer.observe(targetNode, config)`

26. Dynamic Content and Templates

- **Creating a Template Literal**: ``const template = `<div>${variable}</div>` ``
- **Inserting Dynamic HTML**: `element.innerHTML = template`

27. Performance Monitoring and Debugging

- **Measuring Execution Time**: `console.time('timer'); /* code to measure */ console.timeEnd('timer')`
- **Using Performance API**: `performance.mark('start'); /* code */ performance.mark('end'); performance.measure('My Measure', 'start', 'end')`

28. Mobile and Responsive Design

- **Detecting Mobile Device**: `if(/Android|webOS|iPhone|iPad|iPod|BlackBerry/i.test(navigator.userAgent)) { /* mobile specific code */ }`
- **Handling Orientation Change**: `window.addEventListener('orientationchange', handleOrientationChange)`

29. Advanced Networking and Streams

- **Using Fetch with Streams**: `fetch('url').then(response => response.body.getReader().read().then(console.log))`
- **Sending Streams to Server**: `fetch('url', { method: 'POST', body: stream })`

30. Using Web Workers for Background Tasks

- **Creating a Web Worker**: `let worker = new Worker('worker.js')`
- **Sending Message to Worker**: `worker.postMessage('Hello Worker')`
- **Receiving Message from Worker**: `worker.onmessage = function(event) { console.log('Message from worker', event.data) }`

31. Security Measures and Best Practices

- **Content Security Policy**: `meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'"`
- **Cross-Origin Resource Sharing (CORS) Handling**: `Access-Control-Allow-Origin: *`

32. Using Service Workers for Offline Experience

- **Registering a Service Worker**: `if('serviceWorker' in navigator) { navigator.serviceWorker.register('/service-worker.js'); }`
- **Intercepting Fetch Requests in Service Worker**: `self.addEventListener('fetch', event => { event.respondWith(fetch(event.request)); });`
- **Caching Assets for Offline Use**: `caches.open('v1').then(cache => { cache.addAll(['offline.html', 'offline.js']); });`

33. Advanced Browser Features and Detection

- **Detecting Online/Offline Status**: `window.addEventListener('online', onlineHandler); window.addEventListener('offline', offlineHandler);`
- **Feature Detection**: `if('geolocation' in navigator) { navigator.geolocation.getCurrentPosition(position => {}); }`
- **Getting Browser Language**: `const language = navigator.language || navigator.userLanguage;`

34. Working with Documents and Elements

- **Removing All Child Nodes**: `while(element.firstChild) { element.removeChild(element.firstChild); }`
- **Cloning an Element**: `const clone = element.cloneNode(true);`
- **Inserting HTML After an Element**: `element.insertAdjacentHTML('afterend', '<div>New Element</div>');`
- **Scrolling to an Element**: `document.querySelector('#element').scrollIntoView();`
- **Toggling Fullscreen for Element**: `if (element.requestFullscreen) { element.requestFullscreen(); } else if (element.exitFullscreen) { document.exitFullscreen(); }`

35. Web Audio and Video

- **Playing Audio**: `new Audio('file.mp3').play();`
- **Controlling Video Playback**: `const video = document.querySelector('video'); video.play(); video.pause();`

36. Working with Maps and Sets

- **Creating a Map and Adding Items**: `let map = new Map(); map.set('key', 'value');`
- **Retrieving and Deleting in Map**: `map.get('key'); map.delete('key');`
- **Creating and Using Sets**: `let set = new Set([1, 2, 3]); set.add(4); set.has(1); set.delete(4);`

37. Advanced Event Handling

- **Customizing Event Propagation**: `event.stopPropagation(); event.stopImmediatePropagation();`
- **Handling Mouse Events**: `element.addEventListener('mousedown', mouseDownHandler);`
- **Handling Keyboard Events**: `document.addEventListener('keydown', keyDownHandler);`

38. Interactive Web Features

- **Drag and Drop**: `element.setAttribute('draggable', true); element.addEventListener('dragstart', dragStartHandler);`
- **Clipboard Access**: `navigator.clipboard.writeText('Text to copy').then(() => {}, () => {});`
- **Dynamic Script Loading**: `const script = document.createElement('script'); script.src = 'script.js'; document.head.appendChild(script);`

39. Data Manipulation and Computation

- **Data Encryption**: `window.crypto.subtle.encrypt(algorithm, key, data);`
- **Data Decryption**: `window.crypto.subtle.decrypt(algorithm, key, data);`
- **Performing Complex Calculations**: `math.js for complex and matrix calculations`

40. Image and Canvas Manipulation

- **Drawing on Canvas**: `const ctx = canvas.getContext('2d'); ctx.fillRect(10, 10, 150, 100);`
- **Modifying Images with Canvas**: `ctx.drawImage(image, x, y);`
- **Generating Data URL from Canvas**: `canvas.toDataURL('image/png');`

41. Integrating with APIs and SDKs

- **Using Google Maps API**: `new google.maps.Map(document.getElementById('map'), { zoom: 4, center: myLatLng });`
- **Integrating with Social Media APIs**: `FB.api('/me', function(response) { console.log(response); });`

42. Performance Monitoring and Analysis

- **Measuring Performance**: `console.time('process'); /* some process */ console.timeEnd('process');`
- **Using Performance API**: `performance.mark('start'); /* do something */ performance.mark('end'); performance.measure('My Measure', 'start', 'end');`

43. Advanced File Handling

- **Reading Files with FileReader**: `const reader = new FileReader(); reader.onload = function(e) { const text = e.target.result; }; reader.readAsText(file);`
- **Creating and Downloading Files**: `const blob = new Blob(['Hello, world!'], { type: 'text/plain;charset=utf-8' }); saveAs(blob, 'helloWorld.txt');`

44. Real-Time Communication

- **Using WebSockets for Real-Time Communication**: `const socket = new WebSocket('ws://example.com'); socket.onmessage = function(event) { console.log(event.data); };`
- **Implementing WebRTC for Video and Audio**: `const peerConnection = new RTCPeerConnection(configuration);`