Sreenidhi Rajakrishnan
Automation Architect
@sreenidhirajakrishnan

Playwright Coding Last Minute Lookbook

PROGRAM	CONCEPT	CODE
Reverse A String Without .Reverse()	Two-pointer swap converges from both ends toward the middle, mutating in place.	`function reverse(s) { let a=s.split(''), i=0, j=a.length-1; while (i<j) { [a[i],a[j]]=[a[j],a[i]]; i++; j--; } return a.join(''); }`
Check If Two Arrays Have Same Elements	Sorting both normalizes order, then a join comparison checks equality.	`function sameElements(a, b) { if (a.length !== b.length) return false; return [...a].sort().join() === [...b].sort().join(); }`
Write A Function That Retries An API Call 3x	A bounded loop wraps the call in try/catch, rethrowing only on the final attempt.	`async function retry(call) { for (let i=0; i<3; i++) { try { return await call(); } catch (e) { if (i===2) throw e; } } }`
Group An Array Of Objects By A Given Key	reduce() builds a lookup map, bucketing each item by its property value.	`function groupBy(arr, key) { return arr.reduce((acc, o) => { (acc[o[key]] ??= []).push(o); return acc; }, {}); }`
Find The First Duplicate Value In An Array	A Set tracks values seen so far; the first repeat found is the answer.	`function firstDup(arr) { const seen = new Set(); for (const x of arr) { if (seen.has(x)) return x; seen.add(x); } }`
Flatten A Nested Array Without .Flat()	Recursion handles arbitrary depth: concat flattened sub-arrays via reduce().	`function flatten(a) { return a.reduce((r, x) => r.concat(Array.isArray(x) ? flatten(x) : x), []); }`
Debounce A Function Call	A closure holds the timer across calls; each call clears and resets it.	`function debounce(fn, ms) { let t; return (...args) => { clearTimeout(t); t = setTimeout(() => fn(...args), ms); }; }`
Deep-Clone An Object Without A Library	Recursively copies each property, cloning nested objects instead of refs.	`function deepClone(o) { if (o===null || typeof o!=='object') return o; const c = Array.isArray(o) ? [] : {}; for (const k in o) c[k]=deepClone(o[k]); return c; }`
Remove Duplicates From An Array	A Set only keeps unique values; spreading it back dedupes in one line.	`const unique = arr => [...new Set(arr)];`

Sreenidhi Rajakrishnan
Automation Architect
@sreenidhirajakrishnan

Playwright Coding Last Minute Lookbook (cont'd)

PROGRAM	CONCEPT	CODE
Find The Most Frequent Element In An Array	A frequency map counts occurrences, then reduce() picks the highest count.	`function mostFrequent(arr) { const freq = {}; arr.forEach(x => freq[x]=(freq[x]||0)+1); return Object.keys(freq) .reduce((a,b)=>freq[a]>freq[b]?a:b); }`
Check If A String Is A Palindrome	Compare the string to its own reverse — equal means it reads the same both ways.	`const isPalindrome = s => s === [...s].reverse().join('');`
Implement A Sleep/Delay Function	Wrapping setTimeout in a Promise turns a callback API into something awaitable.	`const sleep = ms => new Promise(res => setTimeout(res, ms));`
Chunk An Array Into Groups Of N	Slice repeatedly in fixed steps, pushing each slice into the result array.	`function chunk(a, n) { const out = []; for (let i=0; i<a.length; i+=n) out.push(a.slice(i, i+n)); return out; }`
Write A Custom Promise.All	Track how many promises resolved; resolve the wrapper once all settle, in order.	`function myAll(ps) { return new Promise((resolve, reject) => { let done=0, out=[]; ps.forEach((p,i)=>p.then(v=>{ out[i]=v; if(++done===ps.length) resolve(out);`
Merge Two Sorted Arrays	Two pointers walk both arrays, always taking the smaller head — merge sort's core.	`function merge(a, b) { let i=0, j=0, out=[]; while (i<a.length && j<b.length) out.push(a[i]<=b[j] ? a[i++] : b[j++]); return [...out, ...a.slice(i), ...b.slice(j)];}`
Find The Missing Number In A 1-N Range	The expected sum of 1..n minus the actual sum reveals what's missing.	`function missingNumber(arr) { const n = arr.length + 1; return n*(n+1)/2 - arr.reduce((a,b)=>a+b, 0); }`
Implement Retry-With-Backoff	Same retry loop, but each failed attempt waits exponentially longer.	`async function retryBackoff(call, max=4) { for (let i=0; i<max; i++) { try { return await call(); } catch (e) { await sleep(2**i * 100); } } }`

Sreenidhi Rajakrishnan
Automation Architect
@sreenidhirajakrishnan

Playwright Coding Last Minute Lookbook (cont'd)

PROGRAM	CONCEPT	CODE
Flatten An Object Into Dot-Notation Keys	Recurse through nested keys, building a dotted path for each leaf value.	`function flattenObj(o, p='') { let r = {}; for (const k in o) { const key = p ? ${p}.${k} : k; typeof o[k]==='object' && o[k]!==null ? Object.assign(r, flattenObj(o[k], key)) : r[key]=o[k]; } return r; }`
Count Vowels In A String	Spread the lowercased string into characters and filter against a vowel set.	`const countVowels = s => [...s.toLowerCase()] .filter(c => 'aeiou'.includes(c)).length;`
Implement A Simple LRU-Style Cache	A Map preserves insertion order; re-inserting on access marks it recently used.	`class LRU { constructor(n) { this.n=n; this.map=new Map(); } get(k) { if (!this.map.has(k)) return -1; const v=this.map.get(k); this.map.delete(k); this.map.set(k,v); return v; } }`
Poll Until A Condition Becomes True	A while loop awaits the condition and sleeps between checks.	`async function poll(condition) { while (!(await condition())) await sleep(500); }`
Check If Two Objects Are Deeply Equal	Recursively compares keys and values instead of references.	`function deepEqual(a, b) { if (a === b) return true; if (typeof a!=='object'||typeof b!=='object') return false; return Object.keys(a).length===Object.keys(b).length && Object.keys(a).every(k=>deepEqual(a[k],b[k])); }`
Capitalize Each Word In A Sentence	Split on spaces, uppercase each word's first letter, then rejoin.	`const capitalize = s => s.split(' ') .map(w => w[0].toUpperCase()+w.slice(1)) .join(' ');`
Implement A Simple Event Emitter	Listeners are stored in a map keyed by event name; emit() calls each one.	`class Emitter { on(e, fn) { (this._[e] ??= []).push(fn); } emit(e, ...a) { this._[e]?.forEach(fn => fn(...a)); } }`
Convert Key-Value Pairs Into An Object	Object.fromEntries takes [key, value] pairs and builds a plain object.	`const toObject = pairs => Object.fromEntries(pairs);`