TOSA

Your skills. Your advantage.

Tosa JavaScript

Exam Blueprint

contact@isograd.com

Table of Contents

Introduction

to the Tosa® Exam Blueprint

For Tosa® Assessment and Certification

Tosa® (Test on Software Applications)

Tosa® assessments and certifications are designed to determine a candidate's proficiency level by evaluating their skills in office software and digital tools commonly used in a professional or educational environment.

These tests are specifically developed to validate the professional competencies of candidates looking to enhance their employability (employees, students, job seekers, and individuals undergoing career transitions).

Tosa® assessments and certifications are adaptive tests, developed using scientific methodologies. The scoring is based on Item Response Theory (IRT). The test algorithm adapts to each candidate's response in real time, adjusting the difficulty level of subsequent questions until it precisely determines the candidate's skill level by calculating the upper limit of their competencies. As a result, the tests provide a detailed and unique diagnosis of each candidate's abilities.

The rigor and reliability of Tosa® tests stem from the combination of a mathematical model for analyzing question difficulty and the relevance of the questions selected for each candidate (IRT).

Tosa® Exam Blueprint Objective

This exam blueprint outlines all the skills assessed within the domains and sub-domains of the Tosa® assessment and certification tests for JavaScript.

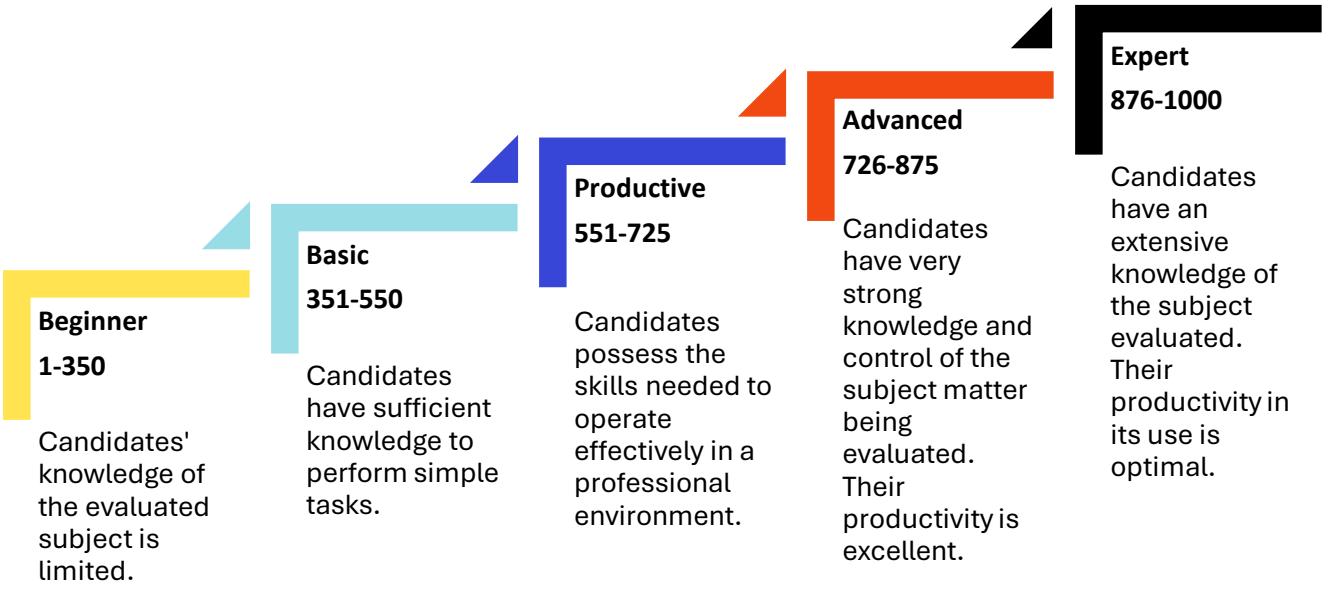
Tosa® assessment and certification solutions are designed to determine learners' proficiency levels using a single scoring scale—ranging from 0 to 1000 for certification—and divided into five levels, from "Beginner" to "Expert," for the assessment.

The purpose of this blueprint is to specify the technical knowledge expected at each level and within each of the four main skill categories of JavaScript. It is intended to help identify the most appropriate teaching or training programs to match a learner's target score.

Unique Tosa® Scoring

The Tosa® assessments and certifications are based on a unique score, divided into five levels.

- Ranging from 1 to 1000 for the certification.
- Divided into five levels, from Beginner to Expert, for the assessment.



Certification failed	**Certification failed**	**Certification earned**	**Certification earned**	**Certification earned**
Certificate of completion issued	Certificate of completion issued	Diploma & Credly digital badge issued	Diploma & Credly digital badge issued	Diploma & Credly digital badge issued

About the Tosa® JavaScript certification

The Tosa JavaScript Certification relies on a database of more than 200 questions. It is composed of 35 questions from the question database and lasts for 1 hour and 30 minutes. The algorithm adapts to each of the candidate's answers to adjust the difficulty level of the questions until reaching the candidate's skill limit. This ensures a precise and accurate result.

Since the test is adaptive, the series of questions that a candidate gets is unique for each test. This algorithm allows for a more accurate evaluation of the candidate's level. It also limits cheating and the memorization of questions on different passages.

Our platform allows individuals to take the certification in a classroom setting, an approved testing center, or remotely via our integrated asynchronous online proctoring solutions.

Our remote proctoring solutions provide added flexibility for both the administrator and the candidate, allowing the certification exam to be taken anywhere, at any time. The candidate only needs an internet connection and a computer equipped with a working webcam and microphone.

Candidates receive a numeric score out of 1000 points. This score corresponds to one of five proficiency levels. Candidates who score between 1 and 550 points do not earn the certification. They receive a certificate of completion in lieu of a diploma. Candidates who score 551 points or above earn the certification and will receive a diploma by email within five business days. If a candidate scores 551 points or above, they will also be eligible to receive a digital Credly badge.

There are no prerequisites to be eligible to take the exam, but to ensure a candidate is well prepared on exam day, we suggest they:

- Take at least one Tosa JavaScript adaptive assessment to estimate their level and get familiar with the test format
- Use the free practice tests on our website for training
- Follow an e-learning or training course (average duration per level is between 10 and 15 hours per certification, so around 150 hours total)

Tosa certification diplomas are valid for three years from the date of issue. This three-year period has been set to ensure that our certifications are consistently accurate and relevant, taking into account software version updates as well as the natural evolution of a candidate's skills over time. Limiting the certification period also reflects the need for life-long learning and continual professional development.

Tosa certifications can be retaken when they expire. Earners willing to improve their score and proficiency level can retake the exam at any time.

Tosa Sequence for Progressive Skills Development

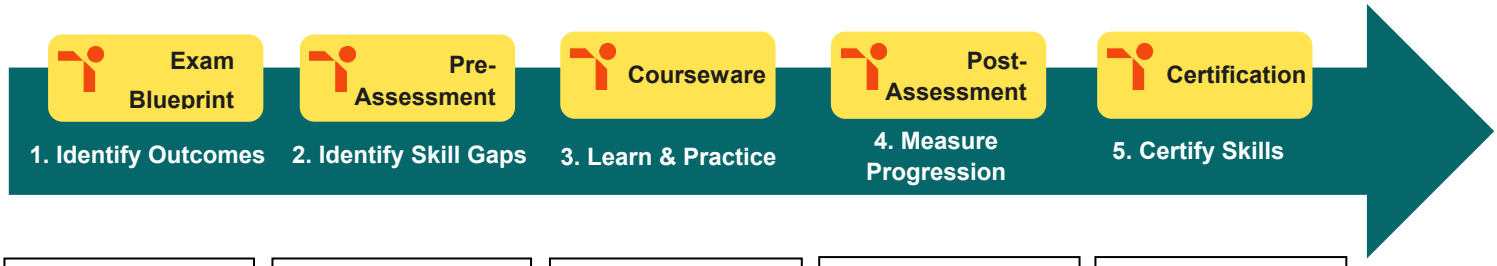
Students and professionals can tailor their certification journey with Tosa through vertical progression, demonstrating increasing proficiency in JavaScript. Starting at a productive level, users can advance to advanced and expert levels as their skills grow. This clear path encourages continuous improvement and validates each stage of their development. Tosa's structure makes it easy to track progress and showcase evolving expertise.

Isograd Learning Lab

The Isograd Learning Lab is a multifaceted and adaptable courseware solution designed to help learners prepare for Tosa certification exams. It offers personalized, self-paced, and fully interactive learning experiences, equipping candidates with the essential digital skills that employers seek. These skills span a wide and varied range, applicable to learners just starting their career journey to those wishing to advance on their path.

The browser-based platform supports all learning styles with a wide array of features, including in-application exercises. These in-application capabilities allow learners to experience real-world examples of specific tasks within a given software environment. The lab includes inclusive learning resources, along with extension activities and project-based learning challenges that foster creativity and critical thinking.

All course content on the Isograd Learning Lab is aligned to a Tosa exam blueprint, which is the foundation of any Tosa certification exam. By aligning course content to an exam blueprint learners will be prepared to take the exam once they complete the courseware.



1. Identify Outcomes	2. Identify Skill Gaps	3. Learn & Practice	4. Measure Progression	5. Certify Skills
Using a Tosa® exam blueprint, instructors integrate Tosa® certifications into their existing content or curriculum and set define outcomes.	Students begin with a formative pre-course assessment to identify their current skills level. Instructors gather data and adjust lesson plans based on students' results.	Instructors match courseware in the Isograd Learning Lab to student's proficiency levels and intended course objectives.	Once the courseware is completed, students take a summative assessment to verify their skills match the intended proficiency level. If skill gaps still exist, instructors can assign additional course content.	When instructors are satisfied with students' progress, certification exams are given.

TOSA
Your skills. Your advantage.

Tosa® JavaScript Level Descriptions

At each level, candidates can:

Beginner

- Demonstrate basic syntax: variables, arithmetic operators, single-line comments.
- Use simple output methods (e.g., console logging).
- Identify and create basic data types: arrays, objects, global objects.
- Locate elements in the DOM using simple selectors..

Basic

- Apply flow control structures: if, else, switch, basic loops.
- Define and reuse simple functions; use let and const.
- Structure and modify data with type conversion, arrays, strings, null/undefined.
- Import modules; interact with APIs like Math and JSON.
- React to basic DOM events and process simple user inputs.
- Debug simple logic and conditional errors.

Productive

- Structure logic with advanced conditionals and arrow functions.
- Write technical documentation with JSDoc.
- Manage complex arrays and escape characters in strings.
- Identify and process variable types dynamically.
- Create reusable modules; manage asynchronous operations with success/error outcomes.
- Perform basic API calls and manipulate DOM content.
- Debug simple code blocks and consult library documentation.

Advanced

- Master complex comparisons (==, ===, abstract equality) and arithmetic on different types.
- Generate formatted strings using template literals.
- Select and manipulate appropriate data structures (arrays, collections, properties).
- Rename module imports; manage multiple asynchronous operations.
- Create, replace, and structure DOM elements dynamically.
- Understand and analyze stack traces for debugging.
- Interpret documentation to resolve standard/error outputs.

Expert

- Perform bitwise operations; manage floating-point precision.
- Master function context switching and advanced error handling.
- Apply prototype-oriented logic and advanced array transformations.
- Capture subexpressions using regex; react to complex DOM events.
- Create robust asynchronous workflows with contextual responses.
- Implement advanced logic, manipulate nested data, and correct multi-layered code issues.

Business Applications

Achievement of Beginner score defines little or limited knowledge of JavaScript, including the language's basic functions and features, highlighting the candidate's inability to perform in a professional environment.	*Junior administrative assistant, content editor, customer support agent, or any position requiring occasional use of JavaScript to manipulate webpage elements, validate form inputs, or automate simple browser-side tasks.*	*Web assistant, junior frontend developer, QA tester, marketing technologist, or any role requiring regular use of JavaScript for scripting reusable components, processing user interactions, or managing basic data flows.*	*Frontend developer, webmaster, integration specialist, digital project manager, or any position requiring advanced JavaScript proficiency to build interactive features, optimize code performance, and debug dynamic applications.*	*Full-stack developer, technical lead, frontend architect, JavaScript consultant, or any role requiring expert-level proficiency in JavaScript for developing complex, scalable web applications, optimizing runtime behavior, and leading development best practices.*

Domain 1: Fundamentals and Applications

Sub-domain: Getting started with JavaScript

Includes identifying when and for whom to use JavaScript, and matching JavaScript to real-world use cases.

Beginner

- Describe what JavaScript is and what it is used for.
- Identify audiences and scenarios where JavaScript is a good choice.
- Recognize basic programming terms and their meanings.

Basic

- Distinguish between types of programming languages.
- Identify the first steps in program development.
- Recognize language-specific features of JavaScript.

Productive

- Identify tasks that JavaScript can be used to accomplish.
- Explain JavaScript's value in applied contexts.
- Identify projects well-suited for JavaScript implementation.

Advanced

- Differentiate JavaScript from other programming languages.

Expert

- Explain optimal and sub-optimal use cases for JavaScript.

Domain 2: Language and Syntax

Sub-domain 1: Mastering basic syntax and control structures

Includes JavaScript variable declarations (let, const), operators, conditionals, and loops for creating decision flows and repetitive tasks.

Beginner

- Identify and use variables (let, const) and basic arithmetic operators.
- Write single-line comments and output data to the console.

Basic

- Apply control structures (if, else, switch) and create simple loops (for, while).
- Use Boolean operators to control flow based on conditions.

Productive

- Configure nested or complex conditionals for workflow control.
- Master loop-related instructions like break, continue, and manage undefined values.

Advanced

- Apply strict and abstract equality comparisons (===, ==).
- Use arithmetic with various types and configure advanced decision trees.

Expert

- Optimize control structures for performance and clarity.
- Handle edge cases involving floating-point precision and bitwise logic.

Sub-domain 2: Defining and using functions

Covers the creation of reusable code with standard and arrow functions, function expressions, parameter handling, and return values.

Beginner

- Create and call simple custom functions.
- Use parameters and return values for basic operations.

Basic

- Structure reusable functions to encapsulate logic.
- Read user input and pass it to functions.

Productive

- Create concise arrow functions for inline logic.
- Write JSDoc comments to document function purpose and usage.
- Group functions into modules for reuse.

Advanced

- Combine functions with conditional logic and loops in structured workflows.
- Refactor functions for clarity, maintainability, and readability.

Expert

- Master function context switching (e.g., callbacks, closures).
- Create asynchronous functions that respond with different data based on outcome.

Sub-domain 3: Handling program inputs, outputs, and errors

Encompasses reading from the console or external sources, writing to the console, using strict comparisons, and managing exceptions via try...catch.

Beginner

- Output content to the console.
- Read static values and display results using basic syntax.

Basic

- Read user input via the console or form elements.
- Apply type conversion (e.g., parseInt, String()).

Productive

- Identify runtime variable types and manage undefined values.
- Implement simple error detection and correct logical issues in small programs.

Advanced

- Catch and trace program errors using browser tools and stack traces.
- Analyze program output to identify and fix issues.

Expert

- Master error handling with try...catch blocks and custom error messages.
- Diagnose and resolve execution errors in complex programs or asynchronous flows.

Domain 3: Data Structures and Objects

Sub-domain 1: Creating and manipulating structured data

Includes arrays, strings, and objects; performing operations like accessing, updating, copying, or transforming data in these structures.

Beginner

- Identify and manipulate basic structures like arrays and objects.
- Read and assign values to simple array or object properties.

Basic

- Create structured data using array and object literals.
- Use array and string methods to access or modify values.
- Apply type conversion on primitives to structure or adapt data.

Productive

- Perform advanced operations on arrays (e.g., mapping, filtering).
- Escape characters within complex strings.
- Create new object references and structure reusable data containers.

Advanced

- Create arrays from multiple data sources.
- Select and combine appropriate data structures (arrays, objects, collections) based on context.

Expert

- Apply advanced array manipulation techniques to transform data efficiently.
- Convert structured arrays into optimized output-ready objects.

Sub-domain 2: Understanding object properties and methods

Focuses on adding, modifying, and accessing properties, distinguishing between value and reference, and using object-specific functions.

Beginner

- Recognize global objects and assign/read basic properties.
- Use dot/bracket notation to manipulate data in simple objects.

Basic

- Create complex properties in objects.
- Use string and array methods to access, update, or process object content.

Productive

- Manage undefined values and handle missing object properties gracefully.
- Dynamically assign or update object content at runtime.

Advanced

- Modify object properties using template literals and conditional logic.
- Choose and organize data in key-value formats for structured processing.

Expert

- Understand and manipulate object behavior using advanced method chaining.
- Design scalable object structures for large-scale applications.

Sub-domain 3: Leveraging object prototypes and collections

Covers prototype-based inheritance, identifying data types at runtime, and working with native methods and collections like Set, Map, and custom prototypes.

Beginner

- N/A

Basic

- Recognize the use of null and undefined to express data absence.
- Begin using simple object copying techniques without prototype manipulation.

Productive

- Detect runtime types and adapt processing accordingly.
- Manage objects that inherit shared behaviors for consistent processing.

Advanced

- Identify collections such as Map, Set, or array-like objects.
- Use object prototypes and shared structures for performance and code reuse.

Expert

- Understand and apply prototype-oriented logic to structure inheritance.
- Extend native object behavior via prototype modification or constructor patterns.

Domain 4: Packages and APIs

Sub-domain 1: Importing and using modules

Covers the use of ECMAScript modules, exporting and importing functions, and managing module namespaces and renaming.

Beginner

- N/A

Basic

- Import basic modules from a local file.
- Use built-in APIs like Math or JSON.

Productive

- Create and export simple modules.
- Group and reuse functions across files or scripts.

Advanced

- Rename imported modules to avoid naming conflicts.
- Manage modular architecture for scalable code organization.

Expert

- Integrate and orchestrate multiple modules dynamically.
- Build complex reusable libraries with custom exports and controlled visibility.

Sub-domain 2: Working with asynchronous operations and APIs

Includes understanding async/await, Promises, managing success/error flows, and using fetch or similar tools to interact with remote servers.

Beginner

- N/A

Basic

- Use simple API methods synchronously (e.g., JSON.stringify).
- Handle basic DOM events triggering data changes.

Productive

- Understand and apply asynchronous mechanisms like Promises.
- Retrieve data from a server using HTTP requests (e.g., fetch).
- Handle both success and error cases in asynchronous operations.

Advanced

- Manage multiple asynchronous operations concurrently.
- Chain async logic with conditional outcomes.

Expert

- Create custom asynchronous functions with tailored return behavior.
- Manage complex async flows, handling parallel execution and cascading errors.

Sub-domain 3: Manipulating the DOM and using built-in APIs

Involves reacting to DOM events, dynamically changing DOM elements, using built-in APIs like Math, JSON, and regular expressions (regex).

Beginner

- Locate and reference simple DOM elements (e.g., via getElementById).

Basic

- Handle basic DOM events (e.g., button click).
- Read or write DOM element values based on user input.

Productive

- Add new elements to the DOM dynamically.
- Serialize and deserialize data using JSON methods.

Advanced

- Replace or modify DOM elements using event responses.
- Create regex patterns to extract text from strings.

Expert

- React to complex, custom DOM events.
- Capture subexpressions using advanced regex and trigger corresponding DOM updates.

Domain 5: Applied JavaScript

Sub-domain 1: Writing and debugging code

Covers implementing specifications, debugging with console tools or stack traces, and identifying syntax and logic errors in code.

Beginner

- N/A

Basic

- Write simple scripts that meet a given set of instructions.
- Identify and correct basic logic errors in loops or conditions.

Productive

- Debug common coding errors using logical analysis.
- Iterate through data structures to produce expected outputs.

Advanced

- Interpret error stack traces in browser tools to identify source of bugs.
- Refactor code for readability and maintainability while preserving logic.

Expert

- Identify and correct multiple complex errors in interconnected code blocks.
- Write clean, robust code for advanced logic flows and team contributions.

Sub-domain 2: Understanding documentation and using resources

Includes reading online docs (e.g., MDN), understanding API behavior, using technical comments like JSDoc to document one's code.

Beginner

- N/A

Basic

- Read simple inline code comments and follow technical specifications.
- Recognize documentation elements like function usage examples.

Productive

- Search for and use documentation for common libraries (e.g., Array.prototype.map).
- Understand basic library usage and code examples from documentation.

Advanced

- Find and interpret documentation related to standard vs. error output.
- Use external resources to learn new methods or functions independently.

Expert

- Navigate complex documentation and extract advanced usage patterns.
- Interpret function contracts and expected behavior in large frameworks.

Sub-domain 3: Applying logic to solve real-world problems

Involves writing code that manipulates arrays and objects according to functional or business goals, processing different data types, and optimizing logic for performance.

Beginner

- N/A

Basic

- Translate clear functional requirements into procedural code.
- Solve straightforward problems involving arrays and conditions.

Productive

- Structure logic around given data to produce a filtered or transformed output.
- Choose appropriate flow structures to match technical requirements.

Advanced

- Implement complex logic based on program state and input combinations.
- Optimize performance of conditionally branching code.

Expert

- Design reusable logic for large-scale features or systems.
- Apply business logic requiring nested conditions, recursion, or asynchronous integration.

TOSA

Your skills. Your advantage.

contact@isograd.com

www.tosa.org