TOSA

Your skills. Your advantage.

Tosa Coding Intelligence

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 Coding Intelligence.

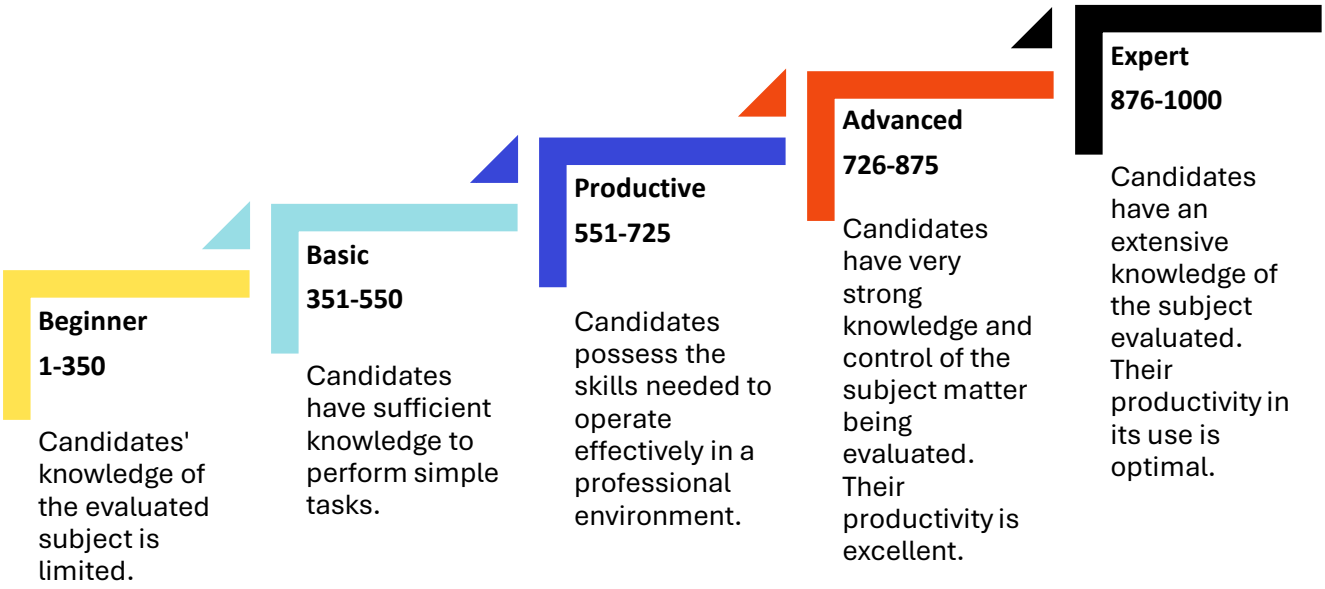
Tosa® assessment and certification solutions are designed to determine learners' proficiency levels using a single scoring scale—ranging from 0 to 1000 for certification—and divided into five levels, from "Beginner" to "Expert," for the assessment.

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 Coding Intelligence. 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® Coding Intelligence certification

The Tosa Coding Intelligence Certification relies on a database of more than 150 questions. It is composed of 35 questions from the question database and lasts for 1 hour. 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 Coding Intelligence 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 including hands-on experience)

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.

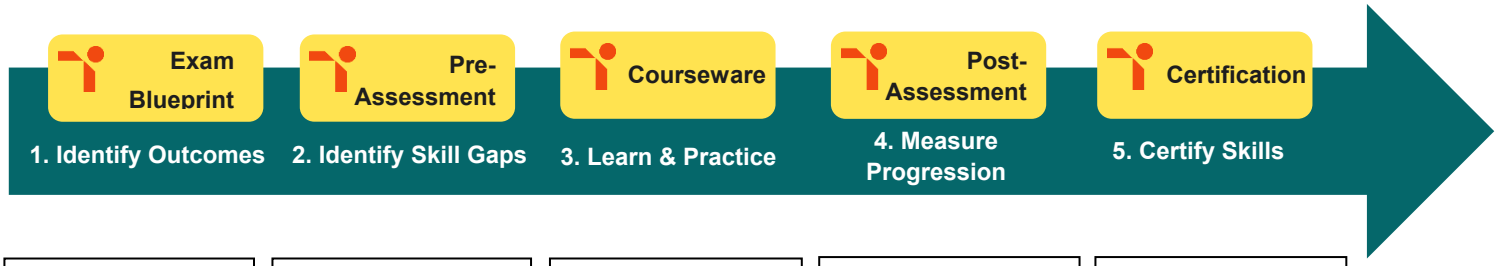
Tosa certifications can be retaken when they expire. Earners willing to improve their score and proficiency level can retake the exam at any time.

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® Coding Intelligence Level Descriptions

At each level, candidates can:

Beginner

- Understand fundamental programming concepts, including how computers execute instructions, process data, and interact with users.
- Read and follow the logic of simple programs, identify common programming elements, and recognize basic coding structures.
- Demonstrate foundational knowledge of software development practices, including testing, debugging, collaboration, and AI-assisted coding tools.
- Understand basic principles of digital citizenship, security, networking, intellectual property, and responsible technology use.

Basic

- Apply core programming concepts to analyze simple programs, identify common logic errors, and create basic reusable code components such as functions.
- Work with variables, data, loops, file storage, and memory concepts to build and understand simple interactive programs.
- Apply software development practices such as debugging, code reviews, version control, and collaborative development workflows.
- Demonstrate responsible coding practices by understanding data protection, intellectual property, algorithmic bias, AI-assisted development, and the ethical use of digital technologies.

Productive

- Design and organize solutions to programming problems by applying algorithmic thinking, decomposition techniques, programming paradigms, and reusable code structures.
- Work effectively with data, files, functions, operators, and data structures while interpreting results and identifying common implementation issues.
- Apply software development practices to improve code quality, including testing, debugging, version control, project planning, and software lifecycle concepts.
- Evaluate the capabilities and limitations of AI-assisted development tools, recognize situations requiring human oversight, and apply responsible, accessible, and ethical coding practices.

Advanced

- Analyze and troubleshoot more complex programming structures, including function interactions, nested control structures, abstraction concepts, and data storage mechanisms.
- Evaluate programming languages, technologies, and data structures to determine their suitability for specific software development scenarios.
- Apply software engineering concepts to project planning, dependency management, and the design of maintainable and scalable solutions.
- Assess privacy, accessibility, fairness, and compliance considerations in software systems and AI-assisted development, while understanding the broader impact of technology on users and society.

Expert

- Analyze and predict the behavior of complex program structures, including recursive execution flows and dynamic data management.
- Evaluate software failures by identifying root causes and proposing effective strategies to prevent similar issues in future projects.
- Critically assess the outputs of AI-assisted development tools by recognizing hallucinations, evaluating prompt quality, and applying appropriate human oversight.
- Assess the ethical implications of algorithms and technology decisions, including their potential impact on users, information access, and society.

Business Applications

Achievement of Beginner score defines little or limited understanding of foundational coding concepts, including basic programming logic, computational thinking, and code interpretation, highlighting the candidate's inability to meaningfully participate in coding-related activities or beginner technical learning environments.	*Student, administrative employee, office support employee, customer service representative, or any individual beginning to explore programming concepts and digital problem-solving. At this level, candidates can understand basic coding principles, follow simple program logic, and recognize common programming structures used in beginner learning environments.*	*Junior technical support employee, digital operations assistant, QA support assistant, STEM student, or any role requiring foundational understanding of programming logic to interact with technical teams, follow structured workflows, interpret simple code, or support basic automation and digital processes.*	*Junior technical professional, project coordinator working with software teams, technical implementation assistant, business analyst support role, or any position requiring the ability to understand coding concepts, analyze simple program behavior, identify common issues, and contribute meaningfully to technical discussions involving software development processes.*	*Achievement of Expert score demonstrates a strong understanding of foundational programming principles, computational thinking, software development practices, and responsible AI-assisted coding concepts. Candidates at this level are well prepared to pursue further technical learning, transition into language-specific programming certifications, or collaborate effectively in environments involving software and digital product development.*

Domain 1: Core Programming Concepts

Sub-domain 1: Understanding programming fundamentals

Introduce the idea of programming languages and how computers use them to follow human instructions.

Beginner

- Understanding Computer Code
- Understanding how Computers follow instructions
- Identifying Applications of Computer code
- Understanding Computer Hardware and Software

Basic

- Differentiating Compiled and Interpreted Code
- Understanding Programming Concepts
- Understanding the Relationship between Programs and Memory
- Identifying File System Terminology and Concepts
- Understanding File Persistence and Data Storage

Productive

- Understanding Programming Paradigms
- Understanding Programming Language Classifications
- Understanding Code Operators
- Manipulating Files with Code
- Completing the Input Function Call

Advanced

- Identifying Compiled Programming Languages
- Identifying High-Level Programming Languages
- Identifying Front-end Programming Languages for User-Facing Website Components
- Identifying where persistent data storage is used in applications

Expert

- Understanding Variable Storage for Dynamic Game Data

Sub-domain 2: Applying computational thinking and code analysis

Learn to observe and predict what a short program or visual code sequence will do.

Beginner

- Following Code Logic
- Tracing Code Logic
- Recognizing Patterns in Programming Problems

Basic

- Identifying Logic Errors in Conditional Statements for Even/Odd Number Checking
- Displaying Function Results as Output
- Understanding Computational Principles

Productive

- Applying Decomposition to Programming Problems
- Understanding Properties of Algorithms
- Identifying Function Call Errors with Incorrect Arguments

Advanced

- Identifying Key Concepts of Abstraction in APIs and Software Design

Expert

- Tracing Recursive Function Execution and Output
- Identifying the Correct Pseudocode Algorithm for Finding the Second Largest Number in an Unsorted List
- Identifying the Most Efficient Algorithm for Finding Maximum Values in Lists

Sub-domain 3: Writing and improving programs

Write and correct short sequences of code using visual or beginner-friendly programming tools.

Beginner

- Understanding the Purpose of Code Comments
- Identifying Event Handlers in Response to User Actions
- Handling User Input with Code

Basic

- Understanding Loop Execution and Output Sequencing
- Creating a Simple Function
- Fixing Issues with Wrong Output

Productive

- Choosing an Appropriate Programming Paradigm
- Identifying correct indentation for conditional statement execution

Advanced

- Completing Nested Loops to Generate Ordered Pairs

Expert

- Recognizing Code Refactoring Opportunities Through Design Patterns and Abstraction

Domain 2: Modular and Structural Programming

Sub-domain 1: Building with functions and reusable code

Create and organize reusable sections of code to structure programs clearly and improve readability.

Beginner

- Understanding the purpose of code reusability and modularity
- Understanding Benefits of Programming Style Guides
- Understanding comments in programming
- Understanding multi-line comments
- Understanding Indentation in Programming

Basic

- Using appropriate naming for variables
- Identifying Functional Groups
- Using Programming Libraries
- Identifying Key Function Concepts and Definitions
- Passing Parameters to a Function
- Identifying best practices for improving code readability through meaningful variable naming

Productive

- Reducing Code Duplication
- Identifying the most effective approach to organize related code into reusable functions
- Understanding Best Practices for Code Comments
- Understanding modular programming concepts and code organization
- Identifying Indentation Rules and Their Impact on Code Structure

Advanced

- Distinguishing Between Parameters and Arguments in Function Definitions

Expert

- Tracing Function Calls and Loop Execution to Determine Output Values

Sub-domain 2: Working with variables and data

Understand how data is represented, stored, and manipulated in programs to make them interactive and meaningful.

Beginner

- Understanding Variable Types and Constants
- Understanding lists
- Using Programming Style Guides
- Matching Values to Data Types
- Identifying the Appropriate Data Type for Boolean Values

Basic

- Identifying appropriate variable types
- Displaying a list and its contents
- Displaying information from data collected in the program
- Storing and Accessing data in a program
- Identifying Type Casting as Data Type Conversion

Productive

- Understanding List Properties and Behavior
- Identifying the Appropriate Data Structure for Key-Value Lookups
- Interpreting Data

Advanced

- Identifying the appropriate data structure for a browser back button implementation

Expert

- Recognizing Stack Data Structure Behavior and LIFO Access Patterns
- Identifying Structural Modifications to Convert a Stack Implementation into a Queue
- Identifying the Structural Constraints that Differentiate Trees from General Graphs
- Analyzing Time Complexity of Graph Operations in Data Structures

Domain 3: Software Development Practices

Sub-domain 1: Testing and improving programs

Identify, test, and fix issues in programs through observation, experimentation, and iteration.

Beginner

- Matching Software Testing Levels with Their Descriptions
- Distinguishing Between Functional and Non-Functional Software Testing
- Identifying Types of Programming Errors and Their Descriptions
- Identifying Different Types of Programming Errors and Their Characteristics
- Identifying the Primary Function of a Debugger
- Identifying Good Debugging Practices
- Identifying Non-Debugging Techniques

Basic

- Identifying arithmetic errors in function logic
- Identifying boundary condition errors in conditional statements
- Identifying variable name spelling errors in function calls
- Identifying incorrect loop structure in pseudocode

Productive

- Identifying off-by-one errors in array loop boundaries
- Identifying Runtime Errors in Code with Inconsistent Data Structure Keys
- Identifying Version Control as a Software Change Management Practice
- Identifying key software testing practices and methodologies

Advanced

- Understanding Recursive Function Call Depth and Execution Flow

Expert

- Identifying Infinite Recursion Due to Missing Base Case
- Identifying the Dominant Time Complexity Factor in a Duplicate Removal Algorithm
- Comparing Algorithm Efficiency: Duplicate Removal Using Arrays vs. Sets

Sub-domain 2: Collaborating and managing projects

Work with others and understand the stages involved in creating, improving, and sharing software projects.

Beginner

- Understanding Version Control Systems for Team Collaboration
- Identifying the Purpose and Function of Code Repositories
- Identifying Pull Requests in Version Control Workflows
- Understanding Code Reviews

Basic

- Understanding how Version Control Systems enable collaborative development through branching and merging
- Identifying the Purpose and Process of Code Reviews
- Identifying Best Practices for Code Reviews and Sprint Demonstrations

Productive

- Understanding the Importance of Scope Definition in Software Project Planning
- Identifying the Requirements Gathering Phase in SDLC
- Ordering the Steps of the Software Development Life Cycle

Advanced

- Identifying Dependencies to Plan Task Sequencing in Software Projects

Expert

- Understanding Best Practices for Reducing Merge Conflicts in Collaborative Development
- Identifying Correct Task Sequencing and Project Dependencies in Software Development
- Prioritizing Code Review Tasks by Risk and Impact
- Identifying the appropriate response to critical security vulnerabilities in release planning
- Identifying Best Practices for Code Review in Architectural Changes

Sub-domain 3: Coding with AI assistance

Use AI-powered tools to support coding tasks while maintaining human responsibility for the final program.

Beginner

- Identifying the purpose and capabilities of AI coding tools
- Understanding Appropriate Uses of AI Coding Tools for Beginner Programmers
- Distinguishing AI Tools and Hardware Components
- Classifying AI Tools by Their Primary Purpose: Coding Assistance vs. Other Applications

Basic

- Identifying Common Capabilities and Limitations of AI Coding Tools
- Matching AI Coding Tasks with Appropriate Prompts

Productive

- Identifying Characteristics and Limitations of AI-Generated Code
- Identifying limitations of AI coding tools in software development
- Recognizing tasks where AI coding tools require human oversight and decision-making

Advanced

- Identifying developer responsibilities when using AI-generated code
- Identifying Effective AI Tool Prompts for Debugging Beginner Programs

Expert

- Recognizing AI Hallucinations
- Ranking AI Prompts by Specificity for Debugging Code Issues

Domain 4: Ethics and Professional Responsibility

Sub-domain 1: Practicing digital citizenship and technology awareness

Act safely, ethically, and responsibly in digital environments while understanding the broader impact of technology on people and society.

Beginner

- Understanding Digital Citizenship in Coding and Technology
- Identifying the Definition and Characteristics of Open-Source Code
- Understanding the Client-Server Model in Network Architecture
- Understanding the Role of TCP/IP Protocols in Internet Communication
- Understanding the Role of Encryption in Securing Internet Communications

Basic

- Identifying Developer Responsibilities in User Data Protection
- Understanding the Distinction Between Open-Source and Public Domain Code
- Understanding Open-Source Licensing and Software Ownership
- Identifying Plagiarism in Code and Understanding Proper Attribution Practices
- Recognizing Plagiarism and Proper Attribution in Code Development
- Identifying Consequences of Personal Data Breaches
- Understanding Data Center Energy Requirements

Productive

- Identifying Dark Patterns in User Interface Design
- Identifying Non-Deceptive User Interface Practices (dark pattern)
- Identifying accessibility barriers in web design for users with color vision deficiencies
- Understanding AI Limitations and the Importance of Human Verification in AI-Generated Code and Content
- Identifying AI Model Limitations and Hallucinations
- Understanding Open-Source Software Licenses and Intellectual Property Rights
- Understanding Copyright and Intellectual Property Rights

Advanced

- Understanding Appropriate Data Breach Response and Regulatory Compliance Procedures
- Identifying Web Accessibility Issues and Violations in User Interface Design
- Identifying Privacy-Respecting Data Collection Practices for Personalized Services

Expert

- Identifying Root Cause Analysis Concepts and Post-Mortem Prevention Strategies
- Understanding Data Privacy and Compliance Requirements for AI Model Training
- Identifying Proper Attribution and Licensing Requirements for Open-Source Code Integration
- Identifying Informed Consent Violations in Healthcare Data Sharing Practices
- Identifying Ethical Violations in Data Privacy and Informed Consent Practices

Sub-domain 2: Applying responsible coding and AI use

Apply coding skills safely, fairly, and responsibly while maintaining human responsibility for code and AI-assisted outputs.

Beginner

- Understanding the Purpose of Input Validation in Software Security
- Identifying Security Risks in Plain Text Password Storage

Basic

- Identifying Sources of Bias in Algorithms and Programs
- Identifying Post-Mortem Review Processes for Root Cause Analysis
- Identifying Best Practices in Software Post-Mortem Processes
- Understanding Blameless Culture in Post-Mortem Incident Reviews

Productive

- Identifying Irresponsible Practices in AI-Assisted Coding
- Identifying Responsible and Irresponsible Uses of AI Tools in Development
- Identifying practices that do not promote accessibility and inclusivity in coding environments
- Understanding Security Principles and Best Practices in Software Development

Advanced

- Identifying Real-World Applications of Algorithms in Society
- Recognizing the Societal Impacts of Algorithms on Information, Employment, and Decision-Making

Expert

- Identifying Ethical Concerns in Algorithm Design and Content Amplification
- Identifying Bias and Fairness Issues in AI Training Data for Hiring Systems
- Understanding Professional Responsibility and Accountability When Using AI Code Generation Tools

TOSA

Your skills. Your advantage.

contact@isograd.com

www.tosa.org