hr | avatar
Illuminating Talent

Test Results and Interview Guide

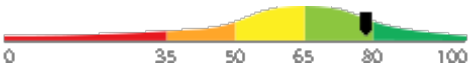
Candidate: **Elizabeth Wantsajob**
Assessment: RUST Programming
Completed: June 27, 2026
Prepared for: Sara Maple
Example Company

What's Included

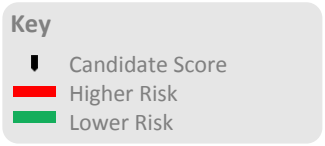
- Overall Score
- Competency Summary Table
- Comparison Matrix
- Detailed Competency Results with Interview Guide

Important Note: The RUST Programming assessment measures key factors related to high performance and tenure in this job. Attribute types measured vary by test, but can include cognitive ability, skills, knowledge, personality characteristics, emotional intelligence, and past behavioral history. This report includes a one page summary, followed by detailed results with an embedded interview guide. Note that these results should always be used as a part of a balanced candidate selection process that includes independent evaluation steps, such as interviews and reference checks.

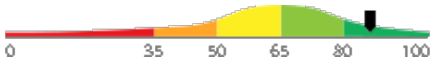
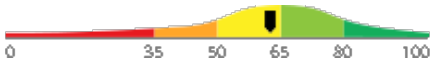
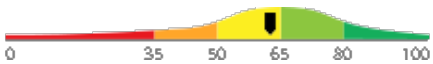
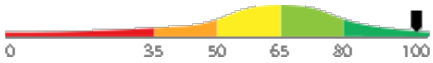
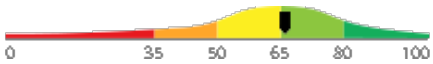
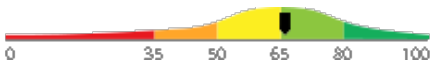
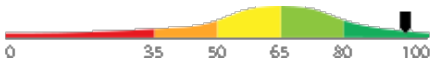
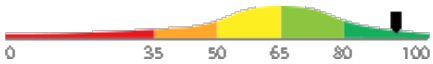
Overall

Candidate	Score	Interpretation
Elizabeth Wantsajob beth.wantsajob@gmail.com RUST Programming June 27, 2026	78	

The candidate demonstrates a solid and competent understanding of Rust programming, including proficiency with core language features such as ownership and borrowing, error handling, pattern matching, traits, and standard library usage. They are likely capable of independently writing, debugging, and maintaining Rust-based business applications with only occasional need for guidance on more advanced or specialized topics. This level of knowledge is consistent with a capable entry-level to mid-level Rust developer.



Competency Summary

Competency	Score	Interpretation
Skills/Knowledge (relates to immediate readiness)		
Cargo, Modules, Crates, and Testing	86	
Ownership, Borrowing, and Lifetimes (Coding Tasks)	62	
Structs, Enums, Traits, and Generics (Coding Tasks)	62	
Collections: Vectors, Hash Maps, and Strings	97	
Error Handling with Result and Option	66	
Ownership, Borrowing, and Lifetimes	66	
Pattern Matching, Control Flow, and Iterators	94	
Structs, Enums, Traits, and Generics	92	

Comparison

Percentile scores indicate how the candidate compares to other test-takers within various groups. The candidate scored equal to or better than the fraction of test-takers indicated by the percentile.

Test-Taker Group	Percentile
Global	78th
North America	65th
United States	65th
Example Company	72nd

Artificial Intelligence (AI) Generated Scores

This table includes one or more scores derived from a large language model AI query. AI-derived scores are non-deterministic. That is, they are not precisely repeatable. Therefore, these scores should always be treated as supplementary information and should never be used exclusively or compared to hard cutoff values.

Estimated Value	Score	Confidence	Interpretation
Knowledge, Skills, and Abilities Summary	-	-	Summary Points (AI): • (Generic Text for Sample Report) Strong performer in Drag and Drop Files tasks, indicating comfort with file management and basic computer interactions. • Demonstrates solid numerical accuracy in Recognizing and Confirming Numbers, a valuable asset in detail-oriented roles. • Moderate overall performance in Analytical Thinking and Attention to Detail, with adequate grammar skills but room for improvement. • Struggles with Reading and Analyzing Problems, which may limit effectiveness in roles requiring critical reading and complex problem-solving. • Lowest performance in Navigating Between Screens, suggesting difficulty with multi-screen software workflows that could impact productivity in computer-intensive roles. Narrative (AI): Elizabeth Wantsajob demonstrates a mixed profile of knowledge, skills, and abilities across the assessed competencies. Elizabeth shows a strong aptitude in Drag and Drop Files, performing well on this technical task and suggesting she is comfortable with this type of computer interaction. This is a notable strength that would translate well into roles requiring file management and basic computer navigation tasks. In the area of Analytical Thinking and Attention to Detail, Elizabeth performs at a moderate level. She demonstrates solid ability in Recognizing and Confirming Numbers, which suggests she is careful and accurate when working with numerical data — a valuable skill in detail-oriented work environments. Her Grammar performance is adequate but leaves room for improvement, indicating she may occasionally make written communication errors. Her weakest area within this competency is Reading and Analyzing Problems, where she struggled to consistently interpret and work through written problem scenarios. This may impact her effectiveness in roles that require critical reading, written comprehension, or complex problem-solving. Elizabeth's most significant area for development is Navigating Between Screens, where she scored considerably lower than the other competencies. This suggests she may have difficulty efficiently moving through software interfaces or multi-screen workflows, which could slow productivity in roles that rely heavily on navigating computer applications or data entry systems. Overall, Elizabeth brings some useful technical strengths, particularly in file management and numerical accuracy, but would benefit from targeted development in software navigation and analytical problem-solving to be fully effective in roles that demand these skills. Computed on: April 2, 2026, 11:09:49PM EDT

Detail

Candidate:	Elizabeth Wantsajob, beth.wantsajob@gmail.com
Assessment:	RUST Programming
Authorized:	June 27, 2026, by Sara Maple, Example Company, qamailsaram.mike@hravatar.com
Started:	June 27, 2026, 8:08:25PM EDT
Completed:	June 27, 2026, 8:08:25PM EDT
Overall Score:	78

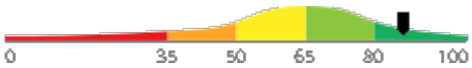
Knowledge and Skills Detail

This section contains a list of job-related knowledge areas and skills that have been evaluated. Low scores in these areas often indicate that additional learning may be required before top performance can be achieved.

Detail | Interview Guide

Cargo, Modules, Crates, and Testing

Score: 86



Description:

Cargo is Rust's build system and package manager, used for every project to manage dependencies, build code, and run tests. Understanding how to organize code into modules and crates, manage Cargo.toml, and write and run unit tests is essential for maintaining any real Rust project.

Interpretation:

Candidate should achieve superior job performance in this area with little or no training.

The candidate exhibits a comprehensive and advanced mastery of Cargo, modules, crates, and testing within Rust. They are well-equipped to independently architect, manage, and test real-world Rust projects, including handling complex dependency management, modular code organization, and robust testing strategies.

How do you write a unit test in Rust, and what do you need to include in your code to make the test runner recognize and execute it?

1	2	3	4	5
Does not know how to write a test or is unaware of the #[test] attribute and the tests module convention.		Knows the #[test] attribute and can write a basic assertion but cannot explain the #[cfg(test)] module or how to run specific tests.		Writes a complete test module with #[cfg(test)], uses assert_eq or assert macros correctly, and explains how to run tests with cargo test.

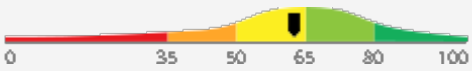
What is Cargo used for in a Rust project, and how do you add an external library as a dependency?

1	2	3	4	5
Cannot explain what Cargo does or does not know how to add a dependency to Cargo.toml.		Knows Cargo builds and runs projects and that dependencies go in Cargo.toml but cannot explain versioning or where to find crates.		Explains Cargo's role, adds a dependency with a version in Cargo.toml, and mentions crates.io as the package registry.

Detail

Ownership, Borrowing, and Lifetimes (Coding Tasks)

Score: 62



Description:

Covers the use of pointers to reference and manipulate memory addresses, along with dynamic memory allocation and deallocation using malloc, calloc, realloc, and free. Includes pointer arithmetic, dereferencing, and avoiding common issues like memory leaks and dangling pointers.

Interpretation:

The candidate exhibits average writing skills, which can hinder high performance in some jobs.

The candidate possesses a moderate working knowledge of C programming, demonstrating familiarity with core concepts including data types, control flow, functions, and basic file I/O. They may require some guidance when working with more advanced topics such as dynamic memory allocation, modular design, or debugging complex logic.

Overall AI Score:	65.0
Lines of Code:	15.0
Syntax Errors:	5.0
AI Confidence Level:	50
Match with Ideal Response (AI):	30.0
Structure:	50.0
Syntax:	30.0

Please see below to view the essay submitted.

Interview Guide

Walk me through how you would dynamically allocate memory for an array of 10 integers, use it, and then properly release it. What issues might arise if you don't follow best practices?

1	2	3	4	5
Cannot write correct allocation code; unaware of free() or memory leak risks.		Writes mostly correct malloc/free code; identifies memory leaks but misses other risks.		Correct malloc, use, and free; identifies leaks, dangling pointers, and NULL check on allocation.

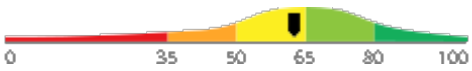
Can you explain what a pointer is in C and describe a situation where you would use one?

1	2	3	4	5
Vague or incorrect definition; cannot describe a practical use case.		Correct basic definition; gives a simple but valid use case with some gaps.		Clear definition with accurate use case; mentions address storage, dereferencing, or dynamic memory.

Detail

Structs, Enums, Traits, and Generics (Coding Tasks)

Score: 62



Description:

Covers the use of pointers to reference and manipulate memory addresses, along with dynamic memory allocation and deallocation using malloc, calloc, realloc, and free. Includes pointer arithmetic, dereferencing, and avoiding common issues like memory leaks and dangling pointers.

Interpretation:

The candidate exhibits average writing skills, which can hinder high performance in some jobs.

The candidate possesses a moderate working knowledge of C programming, demonstrating familiarity with core concepts including data types, control flow, functions, and basic file I/O. They may require some guidance when working with more advanced topics such as dynamic memory allocation, modular design, or debugging complex logic.

Overall AI Score:	65.0
Lines of Code:	15.0
Syntax Errors:	5.0
AI Confidence Level:	50
Match with Ideal Response (AI):	30.0
Structure:	50.0
Syntax:	30.0

Please see below to view the essay submitted.

Interview Guide

Walk me through how you would dynamically allocate memory for an array of 10 integers, use it, and then properly release it. What issues might arise if you don't follow best practices?

1	2	3	4	5
Cannot write correct allocation code; unaware of free() or memory leak risks.		Writes mostly correct malloc/free code; identifies memory leaks but misses other risks.		Correct malloc, use, and free; identifies leaks, dangling pointers, and NULL check on allocation.

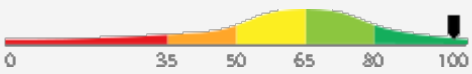
Can you explain what a pointer is in C and describe a situation where you would use one?

1	2	3	4	5
Vague or incorrect definition; cannot describe a practical use case.		Correct basic definition; gives a simple but valid use case with some gaps.		Clear definition with accurate use case; mentions address storage, dereferencing, or dynamic memory.

Detail

Collections: Vectors, Hash Maps, and Strings

Score: 97



Description:

Vectors, hash maps, and strings are the most commonly used data structures in Rust business applications. Knowing how to create, update, iterate over, and safely access these collections is essential for nearly every practical programming task.

Interpretation:

Candidate should achieve superior job performance in this area with little or no training.

The candidate exhibits a strong and comprehensive command of Rust collections, including vectors, hash maps, and strings. They are well-equipped to create, update, iterate over, and safely access these data structures across a wide range of practical and complex Rust programming tasks.

Interview Guide

How would you use a HashMap in Rust to count the number of times each word appears in a list of strings?

1	2	3	4	5
Cannot use HashMap or does not know how to insert or update values; unfamiliar with the entry API.		Inserts and updates values using basic get and insert but is unaware of the entry().or_insert() pattern for cleaner updates.		Uses the entry API fluently, explains why it avoids double lookups, and produces correct, idiomatic Rust code for the word count task.

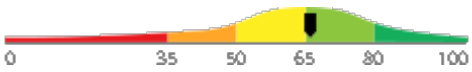
How do you create a vector in Rust, add items to it, and safely access an element by index?

1	2	3	4	5
Cannot create a vector or does not know how to push items; unaware of the difference between indexing and get.		Creates a vector and adds items correctly but does not know about the get method or how to handle out-of-bounds access safely.		Creates a vector, uses push, explains the difference between direct indexing and get, and handles the Option returned by get.

Detail

Error Handling with Result and Option

Score: 66



Description:

Rust does not use exceptions. Instead, it uses the Result and Option types to handle errors and missing values explicitly. Programmers must know how to propagate, match, and handle these types correctly to write reliable, production-ready code.

Interpretation:

Candidate should achieve above average job performance in this area with little or no training.

The candidate demonstrates a solid working knowledge of Result and Option types in Rust, including error propagation and pattern matching in most common scenarios. They are generally capable of writing reliable, production-quality error-handling code with occasional need for guidance on more complex cases.

Interview Guide

How do you use the question mark operator (?) in Rust, and what does it require of the function it is used in?

1	2	3	4	5
Does not know what the ? operator does or confuses it with another operator.		Knows ? propagates errors but cannot explain the return type requirement or how it interacts with From trait conversions.		Accurately explains early return on Err, the requirement for a compatible return type, and how ? simplifies error propagation in real code.

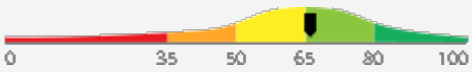
What is the purpose of the Result type in Rust, and how do you check whether a function call succeeded or returned an error?

1	2	3	4	5
Cannot explain Result or confuses it with Option; no mention of Ok or Err variants.		Identifies Ok and Err variants but cannot explain how to handle them beyond using unwrap.		Explains Result clearly, describes match or if let handling, and mentions the risks of using unwrap in production code.

Detail

Ownership, Borrowing, and Lifetimes

Score: 66



Description:

Rust's ownership system is its most distinctive feature and governs how memory is managed without a garbage collector. This includes understanding ownership rules, borrowing (both mutable and immutable references), and lifetime annotations to ensure memory safety in everyday code.

Interpretation:

Candidate should achieve above average job performance in this area with little or no training.

The candidate demonstrates a solid and competent understanding of Rust programming, including ownership and borrowing, error handling with Result and Option types, pattern matching, iterators, and the use of Cargo for dependency management and deployment. They are likely capable of independently writing and maintaining business applications in Rust, though occasional gaps may exist in advanced topics such as concurrency, serialization, or complex trait and lifetime usage. Overall, this candidate is well-suited for entry-level to mid-level Rust development roles.

Interview Guide

Describe a situation where you needed to use a mutable reference in Rust and explain the rules you had to follow to avoid a compile error.

1	2	3	4	5
Cannot recall the rules or confuses mutable and immutable references; no practical example given.		States that only one mutable reference is allowed at a time but cannot fully explain why or give a clear example.		Explains the single mutable reference rule, why it prevents data races, and gives a realistic code scenario.

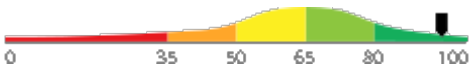
Can you explain what ownership means in Rust and describe what happens to a value when it is moved to a new variable?

1	2	3	4	5
Cannot explain ownership or confuses it with copying; no mention of move semantics.		Explains that the original variable is no longer usable after a move but struggles to explain why.		Clearly explains move semantics, stack vs. heap behavior, and the single-owner rule with a concrete example.

Detail

Pattern Matching, Control Flow, and Iterators

Score: 94



Description:

Rust's match expression, along with if let and while let, provides powerful and safe control flow. Iterators allow programmers to process collections concisely using methods like map, filter, and collect, which are used constantly in everyday Rust code.

Interpretation:

Candidate should achieve superior job performance in this area with little or no training.

The candidate demonstrates an advanced and comprehensive mastery of Rust's pattern matching, control flow, and iterator systems. They can fluently and idiomatically leverage the full breadth of these features—including complex match patterns, iterator chaining, and lazy evaluation—to write safe, concise, and highly efficient Rust code.

Interview Guide

Walk me through how you would use iterator methods to filter a list of numbers and collect only the even ones into a new vector.

1	2	3	4	5
Cannot use iterator methods; would only attempt a manual for loop without awareness of filter or collect.		Uses filter and collect correctly but cannot explain the lazy evaluation of iterators or chaining multiple methods.		Writes a clean iterator chain, explains lazy evaluation, and can discuss chaining map, filter, and collect with correct type annotations.

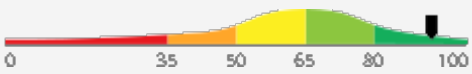
How does a match expression work in Rust, and why is it required to cover all possible cases?

1	2	3	4	5
Cannot explain match or does not know that all cases must be covered; confuses it with a switch statement.		Explains that match must be exhaustive but cannot explain how to use a wildcard or default arm effectively.		Explains exhaustiveness, wildcard arms, destructuring in match arms, and why the compiler enforces complete coverage.

Detail

Structs, Enums, Traits, and Generics

Score: 92



Description:

These are the core tools Rust programmers use to model data and define shared behavior. Structs and enums define custom data types, traits define shared interfaces, and generics allow code to work across multiple types without duplication.

Interpretation:

Candidate should achieve superior job performance in this area with little or no training.

The candidate demonstrates a comprehensive and sophisticated mastery of Rust's data modeling and abstraction mechanisms, including structs, enums, traits, and generics. They are well-equipped to design type-safe, reusable, and idiomatic Rust code across a wide range of complex programming challenges.

Interview Guide

How would you use a trait to define shared behavior across multiple types in Rust, and how does that differ from using a concrete struct method?

1	2	3	4	5
Cannot explain traits or confuses them with structs; no mention of polymorphism or shared interfaces.		Explains that traits define shared methods but struggles to explain how to implement them across types or when to use generics.		Explains trait definitions, implementations on multiple types, and the role of generics or trait objects for flexible, reusable code.

What is the difference between a struct and an enum in Rust, and can you give a simple example of when you would use each?

1	2	3	4	5
Cannot distinguish between structs and enums or provides an incorrect explanation of either.		Correctly distinguishes the two but gives a vague or overly simple example without explaining the choice.		Clearly explains both, gives relevant examples, and articulates why one is preferred over the other in a given situation.

IT Coding Tasks

During the assessment, the candidate was asked to write one or more programs or scripts. Their responses are included below for review.

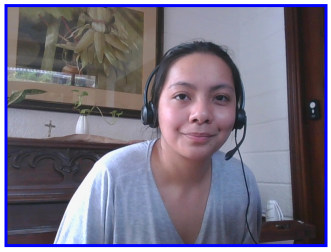
Question or Task	Response

Complete the provided partial C program by filling in the missing sections marked with TODO comments. Your completion must use standard C keywords and libraries.
 	
The program declares a function called duplicate_array that:
1. Takes a const int pointer to a source array and its length as parameters.
2. Uses calloc to allocate a new int array of the same length.
3. Returns NULL if calloc fails.
4. Copies each element from the source array into the new array using pointer arithmetic (not array subscript notation).
5. Returns the pointer to the newly allocated copy.
 	
In main, the program:
1. Declares and initializes a stack array of 4 integers with values 5, 15, 25, 35.
2. Calls duplicate_array to create a heap-allocated copy.
3. Checks for NULL and prints an error and returns 1 if the call failed.
4. Prints each element of the duplicate using a loop.
5. Frees the duplicate array.
 	
Use appropriate indentation, common C coding conventions, and add brief inline comments where needed. Type your completed source code as your response.
	#include <stdio.h>
	#include <stdlib.h>
	
	int *duplicate_array(const int *src, int length) {
	/* TODO: Use calloc to allocate a new array of 'length' integers, return
	NULL if calloc fails, copy elements from src using pointer arithmetic,
	and return the new pointer. */
	calloc(303);
	}
	
	int main(void) {
	/* TODO: Declare and initialize a stack array of 4 integers: 5, 15, 25, 35,
	then call duplicate_array and store the result. Check for NULL and
	print an error message returning 1 if it failed. */
	array[4]={5,15,25,35};
	
	int i;
	
	/* Print each element of the duplicate */
	for (i = 0; i < 4; i++) {
	printf("duplicate[%d] = %d\n", i, *(duplicate + i));
	}
	
	/* Free the duplicate array */
	free(duplicate);
	return 0;
	}
	
	Comments (AI): The code segment has several syntax errors and incomplete implementation. The duplicate_array function does not correctly allocate memory or copy elements. The main function has syntax errors and does not properly call the duplicate_array function. However, the structure and intent of the code are somewhat clear, and the code attempts to follow the requirements.

Identity Confirmation Photos

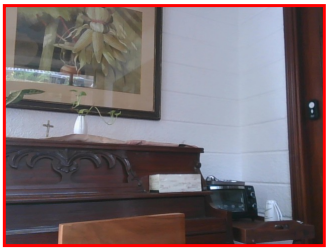
The following photos of the candidate and any identification were uploaded during the assessment session.

Photo Analysis Results	
- Risk:	Medium risk of cheating based on image inconsistencies
- Percent match among processed faces	100%
- Total images processed	17
- Total images with valid faces	14 (82%)
- Total pairs of faces compared	13
- Pairs in which faces matched	13 (100%)

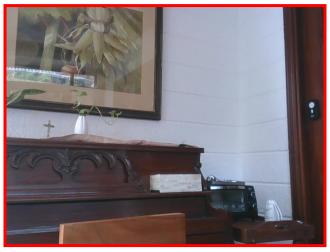


Pre/Post-Test Photo

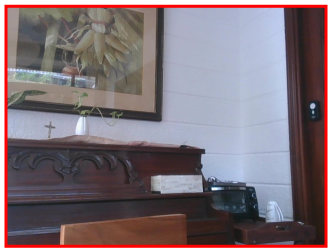
ID Photo



In-Test Error Detected (No Face Detected)

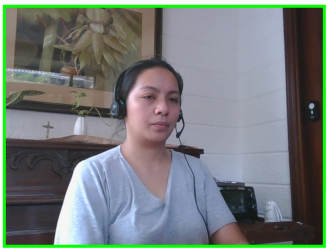


In-Test Error Detected (No Face Detected)

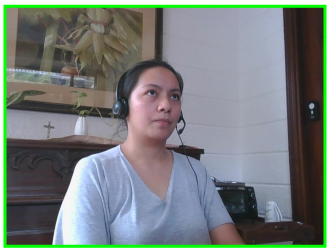


In-Test Error Detected (No Face Detected)

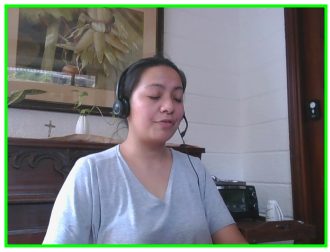
In-Test Photo



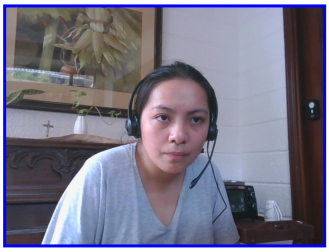
In-Test Photo



In-Test Photo



In-Test Photo



Pre/Post-Test Photo

Resume or CV

Summary Updated on

Motivated career professional with extensive experience in office administration and management. Proven track record of improving efficiency, reducing costs, and enhancing office operations through strategic initiatives and technology implementation.

Objective

I am seeking a role where I can use my many skills and my exceptional judgment and empathy for customers to make a difference to a growing company.

Education

- Associate of Applied Science in Office Administration, Portland Community College, 2020

Experience

- General Office Clerk, Paramount Office Management, 09/2023 – Present
- Administrative Assistant, Global Enterprises Inc., 04/2021 – 08/2023
- Administrative Assistant, Innovative Business Solutions Ltd., 07/2019 – 03/2021

Other Qualifications

- Microsoft Office Specialist (MOS) Certification
- Certified Administrative Professional (CAP)
- International Association of Administrative Professionals (IAAP) Certification

Report Preparation Notes

- Hiring decisions should never be based on a single source of information. The most effective use of this assessment report is as a part of a multi-faceted program of candidate evaluation that includes resume review, interviews, and reference checks.
- Overall vs Percentiles Scores: The overall score reflects the success in the test, based on the mean (average) and standard deviation of the test scores. The percentile score reflects the percentage of test-takers who scored equal or below this overall score. We recommend you use the Overall Score as your primary evaluation criteria. However, percentile scores can often be useful in comparing specific candidates against one another and with a group, such as for test takers in a certain organization or within a certain account.
- Note that comparison information is calculated based on completed instances of this assessment at that time the assessment is scored. As additional instances are completed, the comparative data may change. You can always update a report to the current values by clicking on 'Recalculate Percentiles' within the online results viewing pages at www.hravatar.com.
- Most competency scores are norm-based, which means that they can be interpreted in terms of their distance from the average or mean score. For all scales, a score equal to the mean receives a score of 65 and scores above and below this value are set so that a score change of 15 equals one standard deviation.
- For linear competencies, higher is better across the entire scale. For these scales a score between 65 and 80 (light green) represents 0 to 1 standard deviation above the mean and a score above 80 (dark green) represents more than one standard deviation above the mean. Similarly, a score of 50 - 65 (yellow) represents 0 to 1 standard deviation below the mean, while a score of 35 - 50 (orange) equates to 1 to 2 standard deviations below the mean, and a score below 35 represents more than 2 standard deviations below the mean.
- Sim ID: 20799-1, Key: 0-0, Rpt: 68, Prd: 9621, Created: 2026-06-27 20:08 EDT
- UA: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; Touch; rv:11.0) like Gecko

Score Calculation Detail

The following table provides a summary of how the overall score was calculated from each of the individual competency scores. First, all competency scores are calculated on a scale of 0-100. Note that some competencies use their color category rather than their actual numeric score in the overall calculation. For these, a standard score associated with the assigned color category is used in the overall score calculation rather than the actual numeric score. This is reflected in the ''Score Value Used'' column. Next, a weighted average of scores is computed using individual competency weights, typically set using job analysis data provided by the US Government Occupational Information Network (O*Net).

Competency	Score	How applied to overall	Score Value Used	Weight (%)
Cargo, Modules, Crates, and Testing	86.5094	Numeric Score	86.5094	12.5000
Collections: Vectors, Hash Maps, and Strings	97.3685	Numeric Score	97.3685	12.5000
Error Handling with Result and Option	66.3753	Numeric Score	66.3753	12.5000
Ownership, Borrowing, and Lifetimes	66.5945	Numeric Score	66.5945	12.5000
Ownership, Borrowing, and Lifetimes (Coding Tasks)	62.9784	Numeric Score	62.9784	12.5000
Pattern Matching, Control Flow, and Iterators	94.9996	Numeric Score	94.9996	12.5000
Structs, Enums, Traits, and Generics	92.5830	Numeric Score	92.5830	12.5000
Structs, Enums, Traits, and Generics (Coding Tasks)	62.9784	Numeric Score	62.9784	12.5000
Weighted Average:				78.7984
Final Overall Score:				78

Notes

(This area is intentionally blank - it's reserved as space for your notes.)