hr | avatar
Illuminating Talent

Test Results and Interview Guide

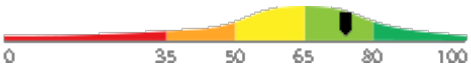
Candidate: **Elizabeth Wantsajob**
Assessment: GO Programming
Completed: June 27, 2026
Prepared for: Sara Maple
Example Company

What's Included

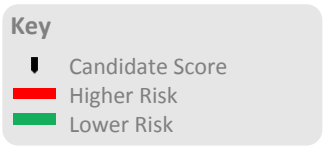
- Overall Score
- Competency Summary Table
- Comparison Matrix
- Detailed Competency Results with Interview Guide

Important Note: The GO 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.

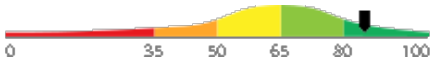
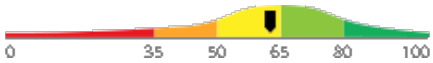
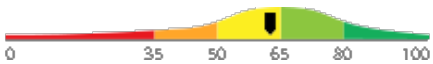
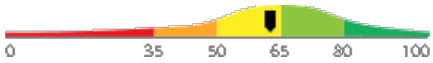
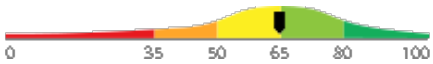
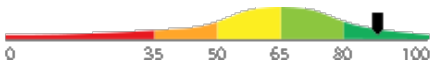
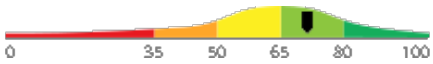
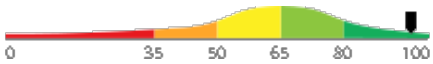
Overall

Candidate	Score	Interpretation
Elizabeth Wantsajob beth.wantsajob@gmail.com GO Programming June 27, 2026	74	

The candidate demonstrates a solid working knowledge of Go programming, including a competent understanding of syntax, data structures, functions, error handling, and standard tooling. Some proficiency in more advanced areas such as concurrency patterns, interfaces, and testing is evident, though mastery of the full breadth of the language may still be developing. This individual is likely capable of contributing effectively in an entry-level to mid-level role with moderate supervision.



Competency Summary

Competency	Score	Interpretation
Skills/Knowledge (relates to immediate readiness)		
Concurrency with Goroutines and Channels	85	
Core Syntax, Data Types, and Control Flow (Coding Tasks)	62	
Functions, Methods, and Interfaces (Coding Tasks)	62	
Core Syntax, Data Types, and Control Flow	62	
Data Structures: Structs, Slices, Maps, and Arrays	65	
Error Handling	88	
Functions, Methods, and Interfaces	71	
Packages, Modules, and Tooling	96	

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	74th
North America	61st
United States	61st
Example Company	68th

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:	GO Programming
Authorized:	June 27, 2026, by Sara Maple, Example Company, qamailsaram.mike@hravatar.com
Started:	June 27, 2026, 8:05:43PM EDT
Completed:	June 27, 2026, 8:05:43PM EDT
Overall Score:	74

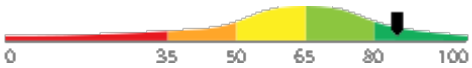
Knowledge and Skills Detail

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

Concurrency with Goroutines and Channels

Score: 85



Description:

Covers Go's built-in concurrency model, including how to launch goroutines, communicate between them using channels, and coordinate work safely. This includes common patterns like using WaitGroups and select statements, which are regularly used in real-world Go services and tools.

Interpretation:

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

The candidate exhibits an advanced and comprehensive understanding of Go's built-in concurrency model, including proficient use of goroutines, channels, WaitGroups, and select statements. They demonstrate the ability to design and implement safe, efficient concurrent systems and are well-equipped to contribute at a high level to real-world Go services and tools.

How do channels work in Go, and what is the difference between a buffered and an unbuffered channel in terms of how they behave when sending and receiving data?

1	2	3	4	5
Cannot explain how channels are used to pass data or distinguish buffered from unbuffered behavior.		Correctly describes the basic send/receive model but gives an incomplete or slightly inaccurate explanation of buffering.		Clearly explains blocking behavior of unbuffered channels, how buffered channels allow sends up to capacity, and gives a use case for each.

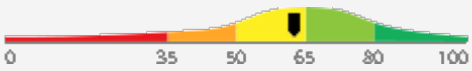
What is a goroutine in Go, and how is starting a goroutine different from starting a thread in a language like Java?

1	2	3	4	5
Cannot describe what a goroutine is or conflates it with OS threads without meaningful distinction.		Correctly identifies goroutines as lightweight and managed by the Go runtime but cannot explain practical implications.		Explains goroutines as lightweight, multiplexed onto OS threads by the scheduler, and notes their low cost enables high concurrency.

Detail

Core Syntax, Data Types, and Control Flow (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.

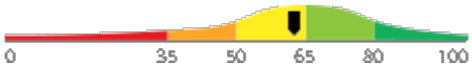
Can you explain what a pointer is in C and describe a situation where you would use one?

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

Functions, Methods, and Interfaces (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.

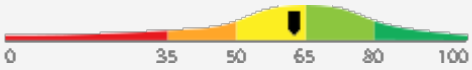
Can you explain what a pointer is in C and describe a situation where you would use one?

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

Core Syntax, Data Types, and Control Flow

Score: 62



Description:

Covers the foundational building blocks of Go programs, including variable and constant declarations, basic data types (int, string, bool, float), operators, and control flow structures such as if/else, for loops, and switch statements. This knowledge is applied in virtually every Go program written or maintained.

Interpretation:

Candidate appears capable of average job performance in this area with little or no training.

The candidate possesses a partial understanding of Go programming fundamentals, including basic syntax, data structures, and standard library usage, but demonstrates inconsistent mastery across key topic areas. Competency in areas such as goroutines, concurrency patterns, dependency management, and testing may require further development to meet entry-level proficiency standards.

Interview Guide

Go only has one looping construct. How do you use the 'for' loop to replicate the behavior of a while loop, and how would you write an infinite loop?

1	2	3	4	5
Unsure Go lacks while/do-while; cannot write a correct infinite loop or condition-based for loop.		Correctly writes both forms but cannot clearly explain why Go was designed this way.		Fluently demonstrates both patterns, explains the single-loop design philosophy, and mentions break/continue usage.

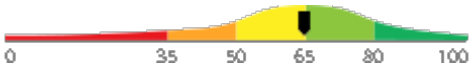
Can you walk me through how you would declare a variable in Go and explain the difference between using 'var' and the ':=' shorthand?

1	2	3	4	5
Cannot explain the difference or confuses syntax; may reference another language's conventions.		Correctly explains both forms but struggles to articulate when to prefer one over the other.		Clearly explains both, notes ':=' is only valid inside functions, and gives a practical example.

Detail

Data Structures: Structs, Slices, Maps, and Arrays

Score: 65



Description:

Covers the primary ways data is organized and stored in Go programs, including fixed-size arrays, dynamic slices, key-value maps, and custom structs. Understanding how to create, access, modify, and pass these structures is essential for nearly all practical Go programming tasks.

Interpretation:

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

The candidate exhibits a solid working knowledge of Go data structures, including the creation, access, modification, and passing of arrays, slices, maps, and structs. They are generally capable of applying these concepts effectively across a broad range of practical Go programming tasks.

Interview Guide

If you have a slice of structs and you want to modify a field on each struct inside a range loop, what is a common mistake developers make and how do you avoid it?

1	2	3	4	5
Unaware of the issue with range loop copies; cannot identify or fix the bug.		Identifies that range produces a copy but solution is incomplete or only partially correct.		Clearly explains that range copies the value, demonstrates using an index or pointer to modify the original, with a correct code example.

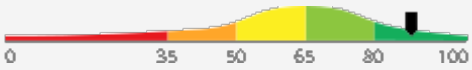
What is the difference between an array and a slice in Go, and why do most Go programmers prefer to use slices?

1	2	3	4	5
Cannot distinguish between the two or believes they are interchangeable.		Correctly identifies that slices are dynamic and arrays are fixed-size but cannot explain the underlying mechanics.		Explains fixed vs. dynamic sizing, describes how slices reference an underlying array, and notes common slice operations like append.

Detail

Error Handling

Score: 88



Description:

Covers Go's explicit approach to handling errors using the built-in 'error' interface, including how to return, check, and wrap errors. Also includes the use of 'panic' and 'recover' for unexpected failures, and the 'defer' statement, which is frequently used alongside error handling and cleanup logic.

Interpretation:

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

The candidate demonstrates a comprehensive and proficient understanding of Go's explicit error handling model, including returning, checking, and wrapping errors, as well as the effective use of panic, recover, and defer. They are well-equipped to write robust, idiomatic Go code that handles both expected and unexpected failure scenarios with confidence.

Interview Guide

What does the 'defer' keyword do in Go, and can you describe a practical situation where using defer is the right choice?

1	2	3	4	5
Cannot accurately describe when deferred functions run or confuses defer with other constructs.		Correctly explains that deferred calls run when the surrounding function returns but gives only a vague or textbook example.		Explains execution order clearly, gives a strong practical example such as closing a file or releasing a lock, and mentions stacking behavior.

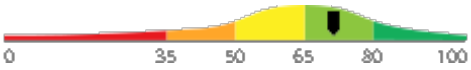
In Go, how is an error typically returned from a function, and how should the calling code check if an error occurred?

1	2	3	4	5
Describes try/catch patterns from other languages; does not know Go returns errors as values.		Correctly describes returning and checking error values but does not mention best practices like checking immediately after the call.		Describes the error-as-value pattern clearly, shows correct if-err-nil check, and mentions wrapping errors with fmt.Errorf or errors.Is/As.

Detail

Functions, Methods, and Interfaces

Score: 71



Description:

Covers how to define and call functions, use multiple return values, work with variadic functions, and attach methods to types. Also includes defining and implementing interfaces, which are central to how Go achieves polymorphism and flexible code design.

Interpretation:

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

The candidate exhibits a solid and competent understanding of Go functions, methods, and interfaces, including the use of multiple return values, variadic functions, and method attachment to types. They are capable of implementing interfaces effectively to support polymorphism and flexible design, with only minor gaps in advanced application.

Interview Guide

How does Go's interface system work, and how does a type 'implement' an interface in Go compared to a language like Java or C#?

1	2	3	4	5
Confuses Go interfaces with those in other languages; believes explicit declaration of implementation is required.		Correctly explains implicit implementation but struggles to explain why this is useful or give a concrete example.		Clearly explains implicit satisfaction, gives a practical example, and articulates the benefit for decoupled, testable code.

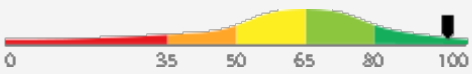
In Go, how do you define a function that returns more than one value, and can you give an example of where that would be useful?

1	2	3	4	5
Cannot write the correct syntax for multiple return values; may not know Go supports this feature.		Writes correct syntax but gives a generic or vague example of practical use.		Writes correct syntax, gives a clear real-world example such as returning a value and an error, and explains the benefit.

Detail

Packages, Modules, and Tooling

Score: 96



Description:

Covers how Go code is organized into packages and modules, how dependencies are managed using 'go mod', and how to use essential Go command-line tools including 'go build', 'go run', 'go fmt', 'go vet', and 'go test'. This knowledge is required for setting up, building, and maintaining any real Go project.

Interpretation:

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

The candidate demonstrates a comprehensive and advanced mastery of Go project organization, module and dependency management, and the full suite of essential Go command-line tools. This individual is well-equipped to independently lead the setup, development, and maintenance of professional-grade Go projects.

Interview Guide

What does 'go vet' do, and how is it different from 'go fmt'? When would you use each one in your development workflow?

1	2	3	4	5
Cannot distinguish between the two tools or describes their purpose inaccurately.		Correctly identifies fmt as a formatter and vet as a static analyzer but gives a vague answer about when to use them.		Clearly explains fmt enforces style and vet catches likely bugs, and describes integrating both into CI pipelines or pre-commit hooks.

What is the purpose of the 'go.mod' file in a Go project, and what command do you use to add a new external dependency?

1	2	3	4	5
Does not know what go.mod is or how Go modules work; may describe an older GOPATH-based workflow.		Knows go.mod tracks dependencies but is unsure of the exact commands or how go.sum relates to it.		Explains go.mod defines the module and its dependencies, uses 'go get' to add packages, and mentions go.sum for integrity verification.

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.

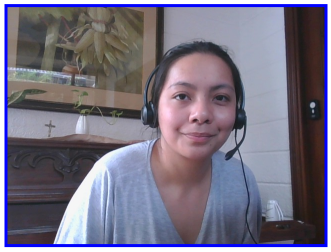
Question or Task	Response

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

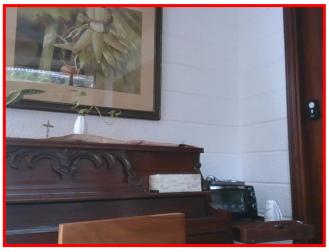
The following photos of the candidate and any identification were uploaded during the assessment session.

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%)

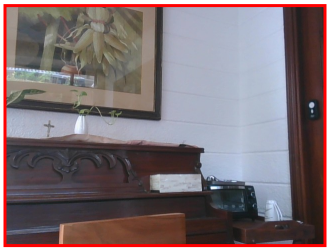


Pre/Post-Test Photo

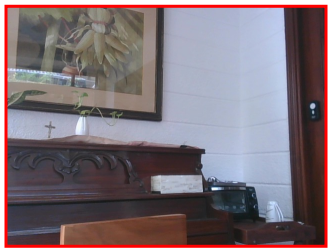
ID Photo



In-Test Error Detected (No Face Detected)

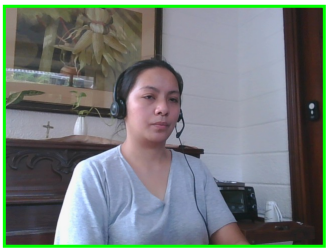


In-Test Error Detected (No Face Detected)

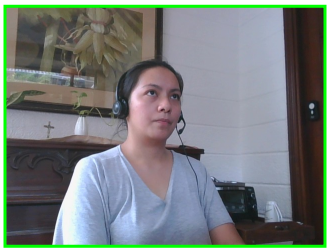


In-Test Error Detected (No Face Detected)

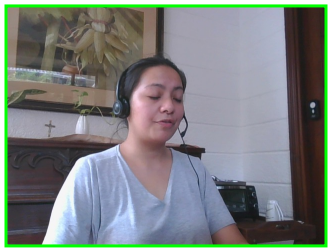
In-Test Photo



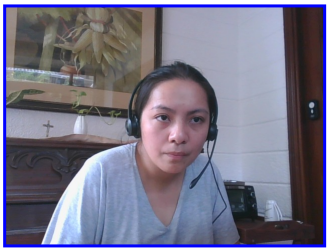
In-Test Photo



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: 20798-1, Key: 0-0, Rpt: 68, Prd: 9620, Created: 2026-06-27 20:05 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 (%)
Concurrency with Goroutines and Channels	85.3201	Numeric Score	85.3201	12.5000
Core Syntax, Data Types, and Control Flow	62.9399	Numeric Score	62.9399	12.5000
Core Syntax, Data Types, and Control Flow (Coding Tasks)	62.9784	Numeric Score	62.9784	12.5000
Data Structures: Structs, Slices, Maps, and Arrays	65.2453	Numeric Score	65.2453	12.5000
Error Handling	88.5778	Numeric Score	88.5778	12.5000
Functions, Methods, and Interfaces	71.7009	Numeric Score	71.7009	12.5000
Functions, Methods, and Interfaces (Coding Tasks)	62.9784	Numeric Score	62.9784	12.5000
Packages, Modules, and Tooling	96.0644	Numeric Score	96.0644	12.5000
Weighted Average:				74.4756
Final Overall Score:				74

Notes

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