hr | avatar
Illuminating Talent

# Test Results and Interview Guide

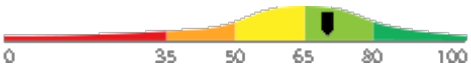
Candidate: **Elizabeth Wantsajob**
Assessment: Ruby Programming (Short)
Completed: June 28, 2026
Prepared for: Sara Maple
Example Company

## What's Included

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

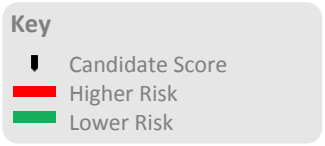
**Important Note:** The Ruby Programming (Short) assessment measures key factors related to high performance and tenure in this job. Attribute types measured vary by test, but can include cognitive ability, skills, knowledge, personality characteristics, emotional intelligence, and past behavioral history. 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

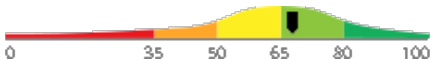
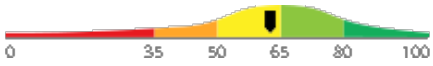
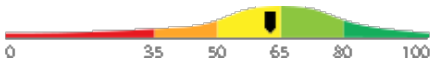
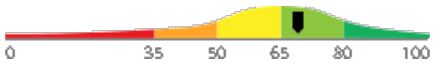
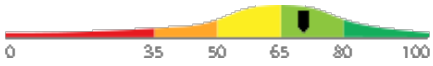
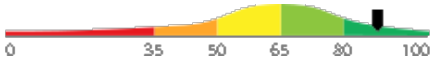
| Candidate | Score | Interpretation |
|---|---|---|
| Elizabeth Wantsajob<br>beth.wantsajob@gmail.com<br>Ruby Programming (Short)<br>June 28, 2026 | 70 |  |



The candidate demonstrates a solid working knowledge of Ruby, including syntax, object-oriented programming, collections, exception handling, and standard tooling, with the ability to write, debug, and maintain Ruby applications at an entry-to-mid level. Some proficiency gaps may exist in advanced areas such as closures, metaprogramming patterns, or comprehensive test-driven development practices. With moderate experience and continued learning, this individual is well-positioned to contribute effectively to Ruby development teams.




## Competency Summary

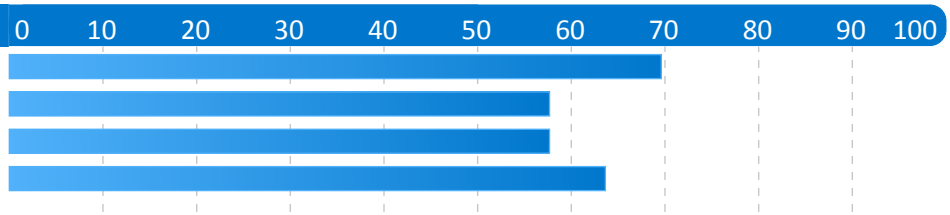
| Competency | Score | Interpretation |
|---|---|---|
| **Skills/Knowledge (relates to immediate readiness)** | | |
| Blocks, Procs, and Lambdas | 68 |  |
| Control Flow, Loops, and Iterators (Coding Tasks) | 62 |  |
| Ruby Syntax, Data Types, and Variables (Coding Tasks) | 62 |  |
| Classes, Objects, Modules, and Inheritance | 69 |  |
| Control Flow, Loops, and Iterators | 70 |  |
| Ruby Syntax, Data Types, and Variables | 88 |  |



## 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 | 70th |
| North America | 58th |
| United States | 58th |
| Example Company | 64th |



## 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):<br>• (Generic Text for Sample Report) Strong performer in Drag and Drop Files tasks, indicating comfort with file management and basic computer interactions.<br>• Demonstrates solid numerical accuracy in Recognizing and Confirming Numbers, a valuable asset in detail-oriented roles.<br>• Moderate overall performance in Analytical Thinking and Attention to Detail, with adequate grammar skills but room for improvement.<br>• Struggles with Reading and Analyzing Problems, which may limit effectiveness in roles requiring critical reading and complex problem-solving.<br>• Lowest performance in Navigating Between Screens, suggesting difficulty with multi-screen software workflows that could impact productivity in computer-intensive roles.<br>Narrative (AI): Elizabeth Wantsajob demonstrates a mixed profile of knowledge, skills, and abilities across the assessed competencies.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br>Computed on: April 2, 2026, 11:09:49PM EDT |

## Detail

| | |
|---|---|
| Candidate: | Elizabeth Wantsajob, beth.wantsajob@gmail.com |
| Assessment: | Ruby Programming (Short) |
| Authorized: | June 28, 2026, by Sara Maple, Example Company, qamailsaram.mike@hravatar.com |
| Started: | June 28, 2026, 8:15:24PM EDT |
| Completed: | June 28, 2026, 8:15:24PM EDT |
| Overall Score: | 70 |

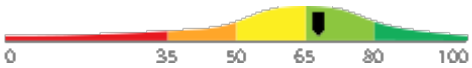
## Knowledge and Skills Detail

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

### Blocks, Procs, and Lambdas

Score: 68




*Description:*

Covers Ruby's approach to passing chunks of reusable code using blocks, Procs, and lambdas. These are core Ruby features used heavily with iterators, callbacks, and custom methods, and understanding them is key to writing clean, idiomatic Ruby.

*Interpretation:*

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

The candidate exhibits a solid understanding of blocks, Procs, and lambdas and can apply them effectively in common Ruby programming scenarios. They are likely comfortable using these constructs with iterators, callbacks, and custom methods, with only occasional gaps in more advanced usage.

What are the key differences between a Proc and a lambda in Ruby, and how do those differences affect how you would use each one in practice?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot identify meaningful differences or conflates Proc and lambda behavior. | | Identifies argument handling or return behavior differences but cannot explain practical impact clearly. | | Accurately explains argument strictness and return behavior differences and gives a concrete use case for each. |

---

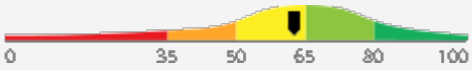
Can you explain what a block is in Ruby and give an example of how you would pass a block to a method and use it inside that method?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot explain what a block is or how to pass and call one inside a method. | | Correctly explains blocks conceptually and shows basic usage but does not demonstrate yield or block_given?. | | Clearly explains blocks, demonstrates yield, and optionally shows block_given? for safe handling with a practical example. |

## Detail

### Control Flow, Loops, and Iterators (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. |

---

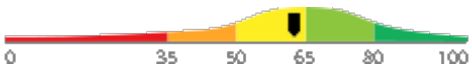
Can you explain what a pointer is in C and describe a situation where you would use one?

| 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

### Ruby Syntax, Data Types, and Variables (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. |

---

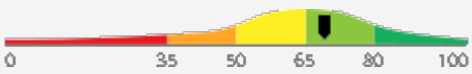
Can you explain what a pointer is in C and describe a situation where you would use one?

| 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

### Classes, Objects, Modules, and Inheritance

Score: 69




*Description:*

Covers how Ruby implements object-oriented programming through class definitions, instance and class methods, inheritance, and the use of modules as mixins. Understanding these concepts is essential for organizing and reusing code in any Ruby application.

*Interpretation:*

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

The candidate exhibits a solid understanding of Ruby's object-oriented programming model, including class and instance methods, inheritance hierarchies, and the use of modules as mixins. Minor gaps in knowledge may exist, but the candidate is generally well-equipped to organize and reuse code effectively in Ruby applications.

## Interview Guide

What is the difference between using inheritance and using a module mixin in Ruby? Can you describe a scenario where you would choose a module over a class hierarchy?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot clearly distinguish inheritance from mixins or gives an impractical or incorrect scenario. | | Correctly explains both concepts but gives only a textbook scenario without strong practical reasoning. | | Explains single inheritance limitation, uses a concrete example like Comparable or Enumerable, and clearly justifies the mixin choice. |

---

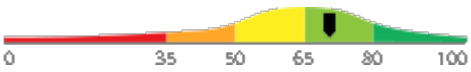
Can you explain what a class is in Ruby and describe how you would create a simple class with an instance variable and a method that uses it?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot define a class correctly or does not understand how instance variables relate to methods. | | Correctly defines a class and instance variable but gives a minimal or incomplete example. | | Clearly defines the class, initializes instance variables in 'initialize', and demonstrates a working method with a practical example. |

## Detail

### Control Flow, Loops, and Iterators

Score: 70




*Description:*

Covers how Ruby controls the execution path of a program using conditionals (if/elsif/else, unless, case/when), loops (while, until, for), and Ruby's built-in iterators (each, map, select, reject, reduce). These constructs are used constantly when writing any meaningful Ruby logic.

*Interpretation:*

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

The candidate shows a solid command of Ruby control flow, loops, and iterators, and is capable of applying conditionals, loops, and a range of built-in iterators confidently in most programming contexts. Minor gaps in knowledge or edge-case handling may exist, but overall they are well-equipped to write meaningful and functional Ruby logic.

## Interview Guide

Explain the difference between 'map', 'select', and 'reduce' in Ruby. How would you decide which one to use when processing a collection?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Confuses the return values or purposes of these methods; cannot give a clear decision rule. | | Correctly explains each method in isolation but struggles to articulate a clear decision framework. | | Fluently explains transformation vs. filtering vs. accumulation, with practical examples and a clear decision rationale. |

---

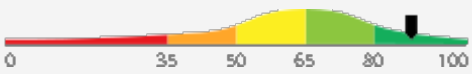
Can you describe how you would use an 'if' statement and an 'unless' statement in Ruby, and explain when you might prefer one over the other?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot explain unless or confuses its logic with if; unable to describe a preference. | | Correctly explains both but gives a generic or unclear rationale for choosing between them. | | Clearly explains the logical inversion of unless and gives a readable, idiomatic example of when unless improves code clarity. |

## Detail

### Ruby Syntax, Data Types, and Variables

Score: 88




*Description:*

Covers the foundational building blocks of Ruby code, including correct use of Ruby syntax, core data types (strings, integers, floats, booleans, nil, symbols), variable types (local, instance, class, global), and operators. This knowledge is applied in virtually every line of Ruby code written or read.

*Interpretation:*

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

The candidate exhibits a comprehensive and advanced understanding of Ruby programming, spanning syntax, object-oriented design, file handling, exception management, testing, and Ruby conventions. They demonstrate the knowledge and skills expected of a proficient mid-level Ruby developer capable of writing, debugging, deploying, and maintaining complex business applications. This individual is well-positioned to contribute effectively to Ruby-based projects with minimal oversight.

## Interview Guide

Walk me through the different types of variables in Ruby — local, instance, class, and global — and explain when you would use each one in a real application.

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Confuses variable types or cannot explain scope differences with practical examples. | | Correctly identifies variable types and their scope but gives only surface-level usage examples. | | Accurately explains scope, naming conventions, and gives clear, practical examples for each variable type in an application context. |

---

Can you explain the difference between a symbol and a string in Ruby, and describe a situation where you would choose one over the other?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot distinguish symbols from strings or gives incorrect explanation of their differences. | | Correctly notes symbols are immutable and memory-efficient but struggles to give a practical use case. | | Clearly explains immutability, memory reuse, and gives a concrete example such as hash keys or method names. |

## IT Coding Tasks

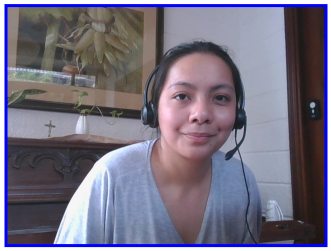
During the assessment, the candidate was asked to write one or more programs or scripts. Their responses are included below for review.

| 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.<br><br>The program declares a function called duplicate_array that:<br>1. Takes a const int pointer to a source array and its length as parameters.<br>2. Uses calloc to allocate a new int array of the same length.<br>3. Returns NULL if calloc fails.<br>4. Copies each element from the source array into the new array using pointer arithmetic (not array subscript notation).<br>5. Returns the pointer to the newly allocated copy.<br><br>In main, the program:<br>1. Declares and initializes a stack array of 4 integers with values 5, 15, 25, 35.<br>2. Calls duplicate_array to create a heap-allocated copy.<br>3. Checks for NULL and prints an error and returns 1 if the call failed.<br>4. Prints each element of the duplicate using a loop.<br>5. Frees the duplicate array.<br><br>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><br>#include <stdlib.h><br><br>int *duplicate_array(const int *src, int length) {<br>/* TODO: Use calloc to allocate a new array of 'length' integers, return<br>NULL if calloc fails, copy elements from src using pointer arithmetic,<br>and return the new pointer. */<br>calloc(303);<br>}<br><br>int main(void) {<br>/* TODO: Declare and initialize a stack array of 4 integers: 5, 15, 25, 35,<br>then call duplicate_array and store the result. Check for NULL and<br>print an error message returning 1 if it failed. */<br>array[4]={5,15,25,35};<br><br>int i;<br><br>/* Print each element of the duplicate */<br>for (i = 0; i < 4; i++) {<br>printf("duplicate[%d] = %d\n", i, *(duplicate + i));<br>}<br><br>/* Free the duplicate array */<br>free(duplicate);<br>return 0;<br>}<br><br>**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

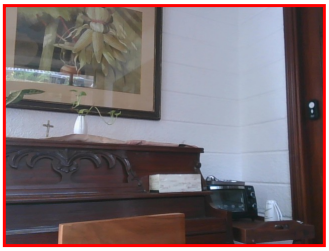
The following photos of the candidate and any identification were uploaded during the assessment session.

| 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%) |

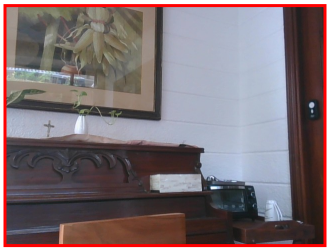


*Pre/Post-Test Photo*

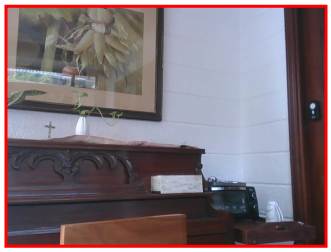
*ID Photo*



*In-Test Error Detected (No Face Detected)*

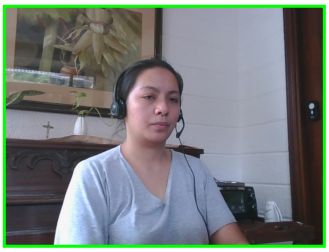


*In-Test Error Detected (No Face Detected)*

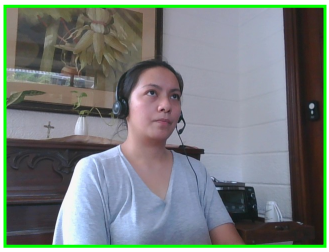


*In-Test Error Detected (No Face Detected)*

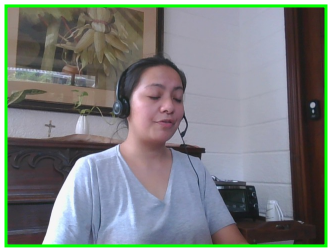
*In-Test Photo*



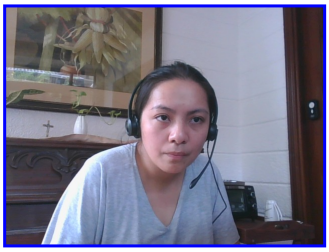
*In-Test Photo*



*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: 20827-1, Key: 0-0, Rpt: 68, Prd: 9649, Created: 2026-06-28 20:15 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 (%) |
|---|---|---|---|---|
| Blocks, Procs, and Lambdas | 68.2669 | Numeric Score | 68.2669 | 16.6667 |
| Classes, Objects, Modules, and Inheritance | 69.5550 | Numeric Score | 69.5550 | 16.6667 |
| Control Flow, Loops, and Iterators | 70.7674 | Numeric Score | 70.7674 | 16.6667 |
| Control Flow, Loops, and Iterators (Coding Tasks) | 62.9784 | Numeric Score | 62.9784 | 16.6667 |
| Ruby Syntax, Data Types, and Variables | 88.4589 | Numeric Score | 88.4589 | 16.6667 |
| Ruby Syntax, Data Types, and Variables (Coding Tasks) | 62.9784 | Numeric Score | 62.9784 | 16.6667 |
| Weighted Average: | | | | 70.5008 |
| Final Overall Score: | | | | 70 |

## Notes

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