hr | avatar
Illuminating Talent

# Test Results and Interview Guide

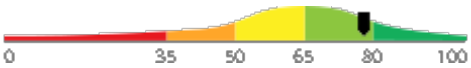
Candidate: **Elizabeth Wantsajob**
Assessment: Swift Programming (Short)
Completed: June 28, 2026
Prepared for: Sara Maple
Example Company

## What's Included

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

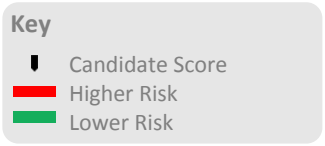
**Important Note:** The Swift Programming (Short) assessment measures key factors related to high performance and tenure in this job. Attribute types measured vary by test, but can include cognitive ability, skills, knowledge, personality characteristics, emotional intelligence, and past behavioral history.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

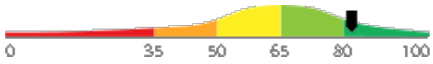
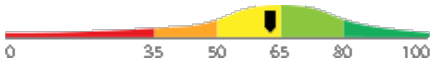
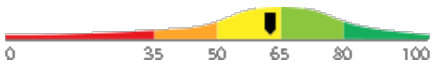
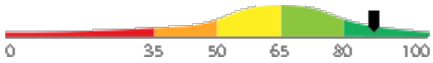
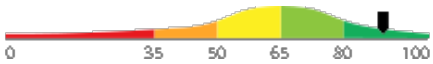
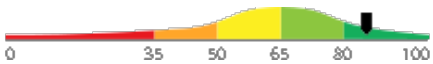
| Candidate | Score | Interpretation |
|---|---|---|
| Elizabeth Wantsajob<br>beth.wantsajob@gmail.com<br>Swift Programming (Short)<br>June 28, 2026 | 78 |  |



The candidate demonstrates a solid, working knowledge of Swift programming across most core areas, including syntax, object-oriented design, optionals, UIKit, networking, and basic concurrency patterns. Some proficiency gaps may exist in advanced topics such as SwiftUI state management, complex memory management scenarios, or performance profiling. This level of knowledge is consistent with a capable entry-level to mid-level iOS or macOS developer who can contribute effectively to real-world projects with moderate supervision.




## Competency Summary

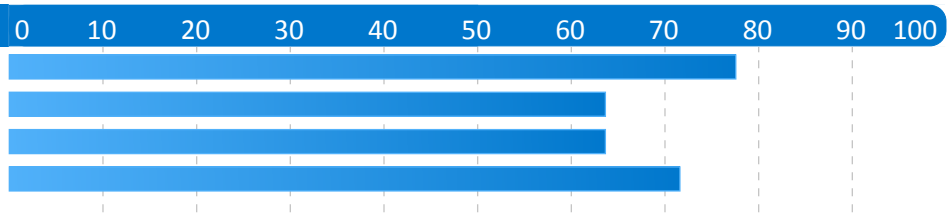
| Competency | Score | Interpretation |
|---|---|---|
| **Skills/Knowledge (relates to immediate readiness)** | | |
| Functions, Closures, and Error Handling | 82 |  |
| Optionals and Safe Value Handling (Coding Tasks) | 62 |  |
| Swift Syntax, Data Types, and Control Flow (Coding Tasks) | 62 |  |
| Object-Oriented and Protocol-Oriented Design | 87 |  |
| Optionals and Safe Value Handling | 89 |  |
| Swift Syntax, Data Types, and Control Flow | 85 |  |



## 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 | 64th |
| United States | 64th |
| 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):<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: | Swift Programming (Short) |
| Authorized: | June 28, 2026, by Sara Maple, Example Company, qamailsaram.mike@hravatar.com |
| Started: | June 28, 2026, 7:05:38PM EDT |
| Completed: | June 28, 2026, 7:05:38PM EDT |
| Overall Score: | 78 |

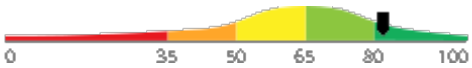
## Knowledge and Skills Detail

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

### Functions, Closures, and Error Handling

Score: 82




*Description:*

Covers how to define and call functions with parameters and return types, write and use closures as first-class values, and handle errors using Swift's throw/catch/try pattern. These skills are used constantly when writing application logic, callbacks, and API interactions.

*Interpretation:*

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

The candidate exhibits an advanced and comprehensive mastery of Swift functions, closures, and error handling. They are highly proficient in writing sophisticated closures, designing robust error handling strategies, and applying these skills fluently across complex application logic, callbacks, and API interactions.

How does Swift's error handling work with do, try, and catch, and how would you decide whether to use try? versus try! versus try?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot explain the do-catch pattern or misuses try!, try?, and try interchangeably. | | Explains do-catch correctly but is uncertain about when to safely use try? or try!. | | Clearly explains all three try forms, discusses safety tradeoffs, and gives appropriate use cases for each. |

---

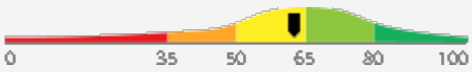
Can you explain what a closure is in Swift and describe a situation where you might use one?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot define closure or confuses it with a regular function with no distinction made. | | Defines closure correctly but can only give a vague or textbook example of usage. | | Defines closure as a self-contained block capturing context, gives practical example like completion handler or sort. |

## Detail

### Optionals and Safe Value Handling (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. |

---

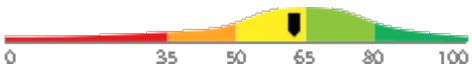
Can you explain what a pointer is in C and describe a situation where you would use one?

| 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

### Swift 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. |

---

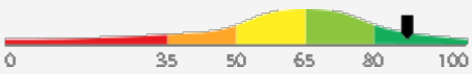
Can you explain what a pointer is in C and describe a situation where you would use one?

| 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

### Object-Oriented and Protocol-Oriented Design

Score: 87




*Description:*

Covers how Swift uses classes, structures, enumerations, protocols, and extensions to organize and structure code. Includes understanding the difference between value types and reference types, protocol conformance, and how extensions add functionality. These concepts are central to writing organized, reusable Swift code.

*Interpretation:*

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

The candidate exhibits an advanced and comprehensive understanding of object-oriented and protocol-oriented design in Swift, demonstrating strong command of classes, structures, enumerations, protocols, and extensions. They can confidently distinguish between value and reference types, apply protocol conformance effectively, and leverage extensions to enhance code functionality and reusability. This level of proficiency reflects the ability to architect well-structured, maintainable Swift code at a high level.

## Interview Guide

How do protocols work in Swift, and how would you use one to make your code more flexible and reusable?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Vaguely describes protocols as similar to interfaces with no practical application given. | | Correctly defines protocol conformance but struggles to explain how it enables flexible design. | | Explains protocol conformance, protocol-oriented design, use with generics, and gives a concrete reusable code example. |

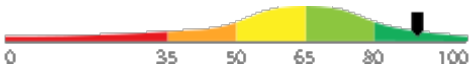
---

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

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot distinguish class from struct or incorrectly states they are interchangeable. | | Identifies reference vs value type difference but gives weak or incorrect usage examples. | | Clearly explains reference vs value semantics, inheritance, and gives practical, appropriate usage examples for each. |

### Optionals and Safe Value Handling

Score: 89




*Description:*

Covers Swift's optional type system, which is used to represent the absence of a value. Includes optional binding (if let, guard let), force unwrapping, nil coalescing, and optional chaining. Safe handling of optionals is a core everyday skill that prevents runtime crashes in Swift applications.

*Interpretation:*

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

The candidate exhibits an advanced and comprehensive mastery of Swift's optional type system and all associated safe value handling techniques. They can confidently and correctly apply optional binding, force unwrapping, nil coalescing, and optional chaining across complex, real-world Swift application contexts, consistently preventing runtime crashes.

When would you use guard let instead of if let to unwrap an optional, and what advantage does it provide?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot distinguish guard let from if let or uses them interchangeably without reason. | | Knows guard exits early but struggles to explain scope or readability benefits clearly. | | Explains early exit pattern, unwrapped value availability in outer scope, and improved code readability. |

---

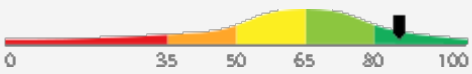
What is an optional in Swift, and what does it mean when a variable is set to nil?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot define optional or confuses nil with zero or an empty string. | | Correctly defines optional and nil but cannot explain how to safely unwrap a value. | | Defines optional clearly, explains nil as absence of value, and mentions safe unwrapping techniques. |

## Detail

### Swift Syntax, Data Types, and Control Flow

Score: 85




*Description:*

Covers the foundational building blocks of Swift, including variables, constants, basic data types (Int, String, Bool, Double), operators, and control flow structures such as if/else, switch, for loops, and while loops. This knowledge is applied in virtually every line of Swift code written.

*Interpretation:*

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

The candidate demonstrates a comprehensive and well-rounded mastery of Swift programming, including advanced language features, design patterns, memory management, concurrency, data persistence, and deployment workflows. Knowledge of both UIKit and SwiftUI, combined with proficiency in debugging, unit testing, and documentation interpretation, indicates strong readiness for mid-level development responsibilities. This score range reflects an individual capable of working independently and contributing at a high level across the full iOS or macOS application development lifecycle.

## Interview Guide

How does Swift's switch statement differ from switch statements in languages like C or Java, and what makes it more powerful?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Describes switch as identical to C/Java with no mention of Swift-specific features. | | Mentions pattern matching or no fall-through but cannot elaborate with examples. | | Discusses no implicit fall-through, pattern matching, ranges, tuples, and where clauses with confidence. |

---

Can you walk me through how you would declare a constant and a variable in Swift, and explain when you would choose one over the other?

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Cannot distinguish let vs var or gives incorrect syntax. | | Correctly explains let vs var but struggles to justify when to use each. | | Clearly explains let vs var, cites safety and compiler optimization as reasons to prefer constants. |

## IT Coding Tasks

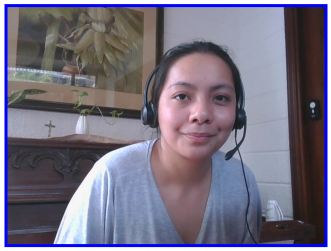
During the assessment, the candidate was asked to write one or more programs or scripts. Their responses are included below for review.

| 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

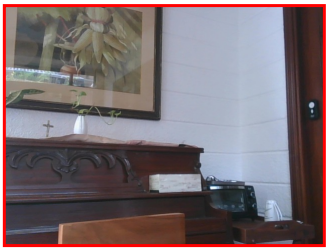
The following photos of the candidate and any identification were uploaded during the assessment session.

| 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%) |

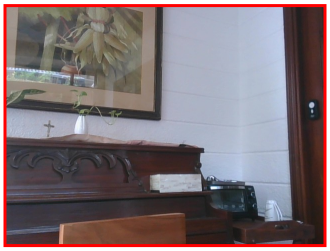


*Pre/Post-Test Photo*

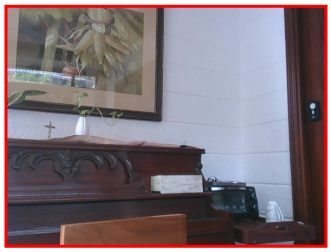
*ID Photo*



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

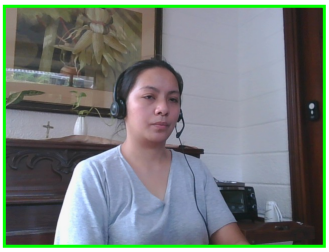


*In-Test Error Detected (No Face Detected)*

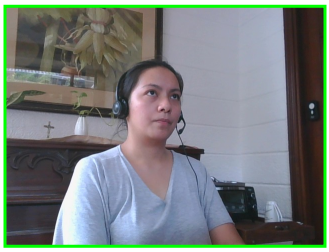


*In-Test Error Detected (No Face Detected)*

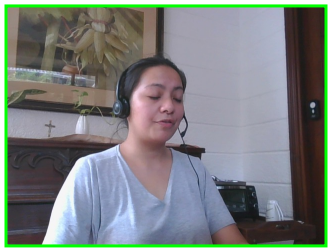
*In-Test Photo*



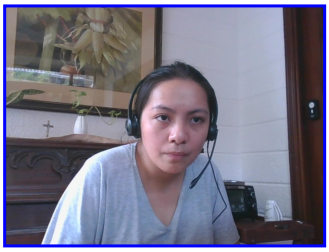
*In-Test Photo*



*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: 20824-1, Key: 0-0, Rpt: 68, Prd: 9646, Created: 2026-06-28 19:05 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 (%) |
|---|---|---|---|---|
| Functions, Closures, and Error Handling | 82.0804 | Numeric Score | 82.0804 | 16.6667 |
| Object-Oriented and Protocol-Oriented Design | 87.4276 | Numeric Score | 87.4276 | 16.6667 |
| Optionals and Safe Value Handling | 89.7114 | Numeric Score | 89.7114 | 16.6667 |
| Optionals and Safe Value Handling (Coding Tasks) | 62.9784 | Numeric Score | 62.9784 | 16.6667 |
| Swift Syntax, Data Types, and Control Flow | 85.7954 | Numeric Score | 85.7954 | 16.6667 |
| Swift Syntax, Data Types, and Control Flow (Coding Tasks) | 62.9784 | Numeric Score | 62.9784 | 16.6667 |
| Weighted Average: | | | | 78.4953 |
| Final Overall Score: | | | | 78 |

## Notes

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