Assignment 2: Syntax, Semantics, and Memory Management

Teja Durga Pavan Kumar Ponneboyina

University of the Cumberlands

2025 Fall - Advanced Programming Languages (MSCS-632-A01) - First Bi-term

Prof. Jay Thom

September 7, 2025

**Part 1 — Analyzing Syntax and Semantics**

**1.1 Section 1 — Syntax errors across Python, JavaScript, and C++**

**Goal:** We need to take three small “sum of array” programs, introduce syntax (and a couple of common runtime) errors, see what each tool reports, and compare behavior (Sebesta, 2018).

**A) Python**

With Intentional error:A white rectangular object with blue text

AI-generated content may be incorrect.

**Mistakes/errors noticed:**

* The missing colon after the function header is caught before running:
  + Typical message: SyntaxError: invalid syntax with a caret pointing at the line end.
* If I fix the colon but keep total = o, Python compiles the file, but at runtime it fails on that line:
  + Message: NameError: name 'o' is not defined.

**Takeaway:** Python catches clear syntax problems at parse time (e.g., missing colon), but some mistakes (like using o instead of 0) surface later as runtime exceptions (Sebesta, 2018; Python Docs, 2025).

Corrected Code:

A white screen with blue text

AI-generated content may be incorrect.

**B) JavaScript**

**Intentional errors**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Mistakes/errors noticed:**

* The space in the function name is a parse-time issue:
  + Typical message: SyntaxError: Unexpected identifier 'Sum'.
* If I fix the name but keep let total = o;, that throws at runtime:
  + ReferenceError: o is not defined.

**Takeaway:** Like Python, JavaScript flags true syntax errors early, but name/lookup problems happen at runtime (MDN, 2025; Sebesta, 2018).

**Corrected JavaScript code**

A white rectangular object with black text

AI-generated content may be incorrect.

**C) C++**

**Intentional errors**

A screenshot of a computer

AI-generated content may be incorrect.

**Mistakes/errors noticed:**

* The compiler fails before running with messages like:
  + error: 'o' was not declared in this scope
* The line with cout << "Sum in C++" " << result << endl; is tricky:
  + Adjacent string literals are concatenated in C++, so this compiles into "Sum in C++ << result", which prints no variable (it’s a logic bug, not a syntax error). Good reminder that some mistakes are legal but wrong (Stroustrup, 2013).

**Takeaway:** C++ is compiled and catches undeclared identifiers and many structural issues at compile time. Some logic problems are still compiled (Sebesta, 2018; Stroustrup, 2013).

**Corrected C++ code**

A white screen with a blue and red text

AI-generated content may be incorrect.

**Quick comparison**

* **Python/JavaScript:** parse-time syntax errors vs. runtime name/lookup errors; dynamic languages often let you run farther before crashing (Sebesta, 2018; Python Docs, 2025; MDN, 2025).
* **C++:** most issues are caught by the compiler early; you get many errors up front (Sebesta, 2018; Stroustrup, 2013).

**1.2 Section 2 — My own programs + semantic analysis**

Here I write tiny programs to highlight **type systems**, **scopes/closures**, and **three semantic differences** that affect behavior/performance.

**Feature: Scope and Closures**

**Python** — function scope, no block scope; nonlocal to mutate outer variables.

A white rectangular object with black text

AI-generated content may be incorrect.

**JavaScript** — block scope with let/const, closures capture lexical environment.

A white rectangular box with a blue and white text

AI-generated content may be incorrect.

**C++** — block scope; closures via lambdas with explicit capture lists.

A white screen with blue text

AI-generated content may be incorrect.

**Three key semantic differences (behavior + performance)**

1. **Typing model**

* **Python/JS:** dynamic typing and late binding; more flexible but many errors appear at runtime; interpreter overhead can add latency (Sebesta, 2018; MDN, 2025; Python Docs, 2025).
* **C++:** static typing and early binding (with templates/overloads), catches more errors at compile time, generally faster code (Sebesta, 2018; Stroustrup, 2013).

1. **Scope & closures**

* **Python:** function scope, no block scope for simple if/for, closures need nonlocal to mutate outer variables (Python Docs, 2025).
* **JS:** block scope with let/const; closures are fundamental and often used for callbacks and modules (MDN, 2025).
* **C++:** block scope is strict since lambdas require explicit captures, the captures affect lifetime and performance (copies vs references) (Stroustrup, 2013).

1. **Coercion & operators**

* **JS** allows implicit coercion ("5" + 2 ⇒ "52"), which can surprise performance and correctness (MDN, 2025).
* **Python** is stricter ("5" + 2 raises TypeError) (Python Docs, 2025).
* **C++** uses overload resolution and implicit conversions, but is still statically typed, because mistakes are mostly caught at compile time (Sebesta, 2018; Stroustrup, 2013).

**Part 2 — Memory Management**

**2.1 Section 3 — Short programs in Rust, Java, and C++**

I would want to show how each language allocates and frees memory, and what happens with errors like leaks or dangling pointers. Rust uses ownership/borrowing (compile-time checks), Java uses garbage collection, and C++ uses manual control (plus RAII smart pointers). Trade-offs are about dev complexity, speed, and safety (Sebesta, 2018; Stroustrup, 2013; Klabnik & Nichols, 2019; Oracle, 2023).

**A) Rust — ownership & borrowing**

A white box with blue text

AI-generated content may be incorrect.

**What it shows:** Rust enforces at compile time: one mutable reference or many immutable ones, but not both at the same time. Use-after-free and double-free are ruled out by the type system (Klabnik & Nichols, 2019).

**Profiling idea**: build with release, we can use tools like valgrind or heaptrack on the binary. Rust won’t leak by default unless you intentionally leak structures.

**B) Java — garbage collection**

A screenshot of a computer

AI-generated content may be incorrect.

**What it shows:** We don’t free memory ourselves; the GC reclaims objects that are no longer reachable. This is easy and safer for developers, but GC can introduce pauses and overhead (Sebesta, 2018; Oracle, 2023).

**Profiling idea:** Use JVisualVM or Java Flight Recorder to monitor heap usage and GC activity.

**C) C++ — manual allocation (and RAII)**

**Raw pointers (easy to get wrong):**

A close-up of a computer screen

AI-generated content may be incorrect.

**RAII with smart pointers (safer):**

**A white box with blue text

AI-generated content may be incorrect.**

**What it shows:** C++ gives you speed and control, but manual new/delete are risky. RAII (e.g., unique\_ptr, shared\_ptr) reduces leaks and dangling pointers by binding lifetime to scope (Stroustrup, 2013; Sebesta, 2018).

**Profiling idea:** Valgrind or AddressSanitizer (-fsanitize=address) to detect leaks and invalid memory access.

**Summary of memory trade-offs**

* ***Easiest to use correctly:*** **Java** (GC does the cleanup).
* ***Fastest potential / most control:*** **C++**, but easiest to misuse without RAII.
* ***Best safety with low runtime cost:*** **Rust**, thanks to ownership/borrowing checked at compile time (Klabnik & Nichols, 2019).

**Choice depends on app type:**

* ***Games/embedded/high-frequency trading:*** often **C++** for max control/speed (with RAII).
* ***Large enterprise/mobile apps:*** often **Java/JVM** for productivity and GC.
* ***Modern systems/networking/security-sensitive services:*** **Rust** for safety and performance (Sebesta, 2018; Stroustrup, 2013; Klabnik & Nichols, 2019; Oracle, 2023).

**References**

* Klabnik, S., & Nichols, C. (2019). *The Rust programming language*. No Starch Press. <https://doc.rust-lang.org/book/>
* MDN Web Docs. (2025). *Closures*; *let*; *Type coercion*. Mozilla. <https://developer.mozilla.org/>
* Oracle. (2023). *Java SE documentation* (Garbage collection overview). <https://docs.oracle.com/javase/>
* Python Documentation. (2025). *Errors and exceptions*; *Scopes and namespaces*. <https://docs.python.org/3/>
* Sebesta, R. W. (2018). *Concepts of programming languages* (12th ed.). Pearson. <https://reader2.yuzu.com/books/9780135102251>
* Stroustrup, B. (2013). *The C++ programming language* (4th ed.). Addison-Wesley. <https://chenweixiang.github.io/docs/The_C++_Programming_Language_4th_Edition_Bjarne_Stroustrup.pdf>