Syntax, Semantics and Memory Management

Mausam Shrestha

GitHub: https://github.com/shrestha-mausam/ucumberlands

Department of Computer Information Sciences, University of the Cumberlands

(MSCS-632) Advanced Programming Languages: Assignment 2

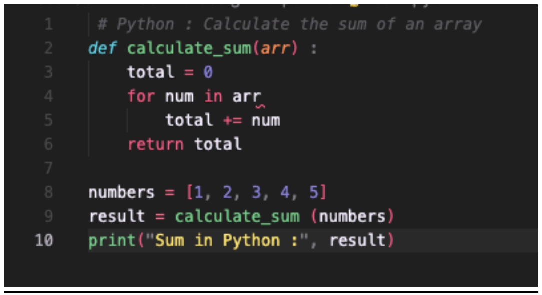
Dr. Vanessa Cooper

May 14, 2025

Part 1: Analyzing Syntax and Semantics

1.1 Syntax Errors

Python

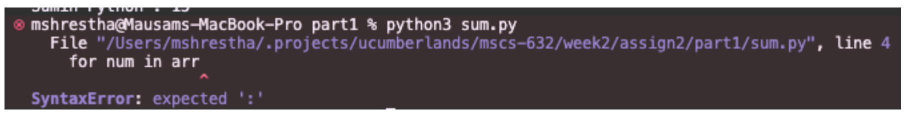


```python
# Python : Calculate the sum of an array
def calculate_sum(arr) :
    total = 0
    for num in arr
        total += num
    return total

numbers = [1, 2, 3, 4, 5]
result = calculate_sum (numbers)
print("Sum in Python :", result)
```

Steps:

- I have removed a semi-colon from line 4 to introduce a syntax error in the given python file.
- I then ran the python file using the Python 3 interpreter using the command:
  - python3 sum.py

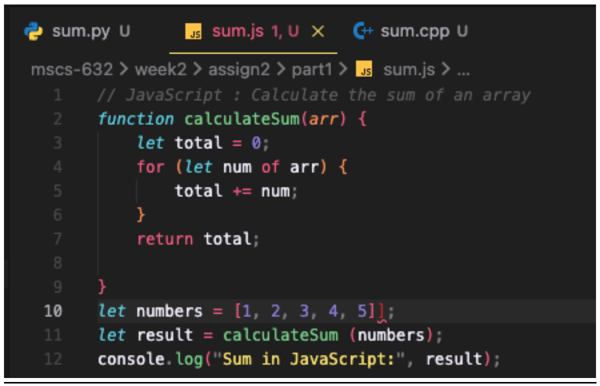
Error handling:



```
mshrestha@Mausams-MacBook-Pro part1 % python3 sum.py
  File "/Users/mshrestha/.projects/ucumberlands/mscs-632/week2/assign2/part1/sum.py", line 4
    for num in arr
                  ^
SyntaxError: expected ':'
```

Python is an interpreted language, so it checks the syntax of the python file before executing it. When we try to run the given python file (sum.py), Python will raise a syntax error informing us that a semi-colon was missing, and in what line, and then it stops the execution before the sum is calculated and the print function is called.

JavaScript

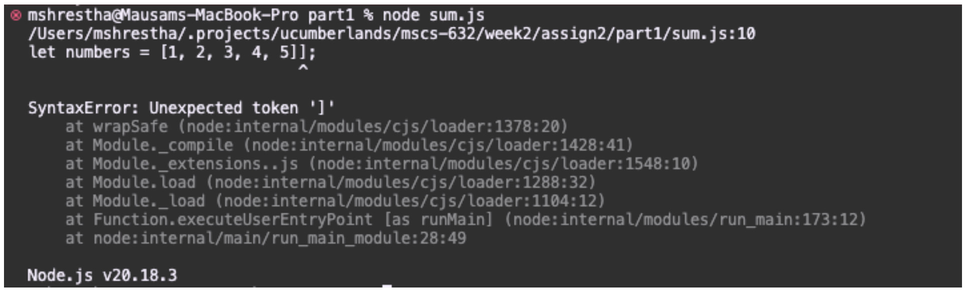


```
1   // JavaScript : Calculate the sum of an array
2   function calculateSum(arr) {
3       let total = 0;
4       for (let num of arr) {
5           total += num;
6       }
7       return total;
8
9   }
10  let numbers = [1, 2, 3, 4, 5]];
11  let result = calculateSum (numbers);
12  console.log("Sum in JavaScript:", result);
```

Steps:

- I added an extra "]" in line 10 to introduce a syntax error.
- I than ran the JavaScript file sum.js using node with the following command:
    - node sum.js

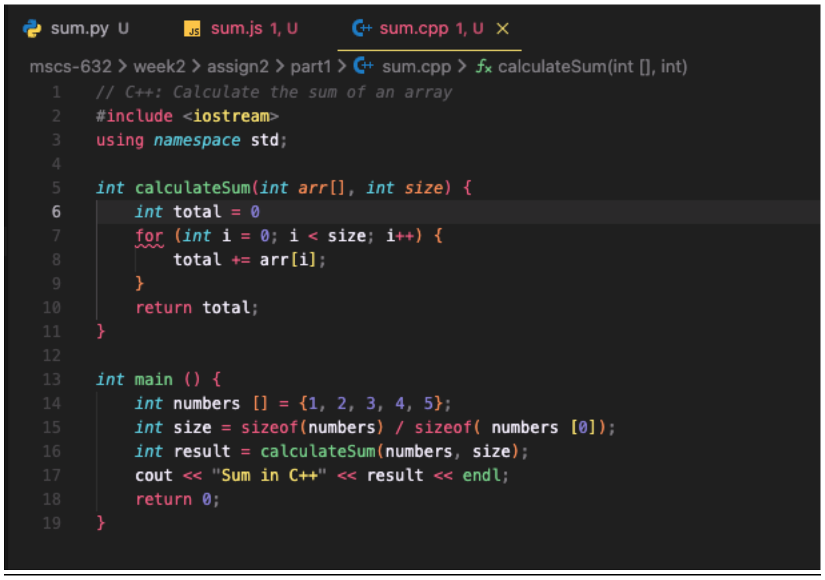
Error handling:



```
mshrestha@Mausams-MacBook-Pro part1 % node sum.js
/Users/mshrestha/.projects/ucumberlands/mscs-632/week2/assign2/part1/sum.js:10
let numbers = [1, 2, 3, 4, 5]];
                             ^

SyntaxError: Unexpected token ']'
    at wrapSafe (node:internal/modules/cjs/loader:1378:20)
    at Module._compile (node:internal/modules/cjs/loader:1428:41)
    at Module._extensions..js (node:internal/modules/cjs/loader:1548:10)
    at Module.load (node:internal/modules/cjs/loader:1288:32)
    at Module._load (node:internal/modules/cjs/loader:1104:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:173:12)
    at node:internal/main/run_main_module:28:49

Node.js v20.18.3
```

JavaScript is also an interpreted language, and the node interpreter catches the syntax error while parsing the file before execution. The node interpreter gives us a more detailed message that includes the line that caused the error, what type of error it is (Syntax Error) , and what modules caused the error while trying to parse the file.

C++



sum.py U   sum.js 1, U   sum.cpp 1, U ×

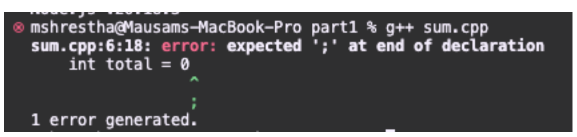
mscs-632 > week2 > assign2 > part1 > sum.cpp > calculateSum(int [], int)

```cpp
1   // C++: Calculate the sum of an array
2   #include <iostream>
3   using namespace std;
4
5   int calculateSum(int arr[], int size) {
6       int total = 0
7       for (int i = 0; i < size; i++) {
8           total += arr[i];
9       }
10      return total;
11  }
12
13  int main () {
14      int numbers [] = {1, 2, 3, 4, 5};
15      int size = sizeof(numbers) / sizeof( numbers [0]);
16      int result = calculateSum(numbers, size);
17      cout << "Sum in C++" << result << endl;
18      return 0;
19  }
```

Steps:

- I removed the semi-colon in line 6 to introduce a syntax error.
- I then compiled the sum.cpp file because c++ files needs to be compiled before it can be executed. I used the following code to compile the sum.cpp file:
    - g++ sum.cpp

Error Handling



```
mshrestha@Mausams-MacBook-Pro part1 % g++ sum.cpp
sum.cpp:6:18: error: expected ';' at end of declaration
    int total = 0
                 ^
                 ;
1 error generated.
```

Unlike, Python and JavaScript, C++ code needs to be compiled. So, when tried to compile the sum.cpp file, the C++ compiler g++ caught our syntax error, printed out what the issue was, and then stop the compilation process.

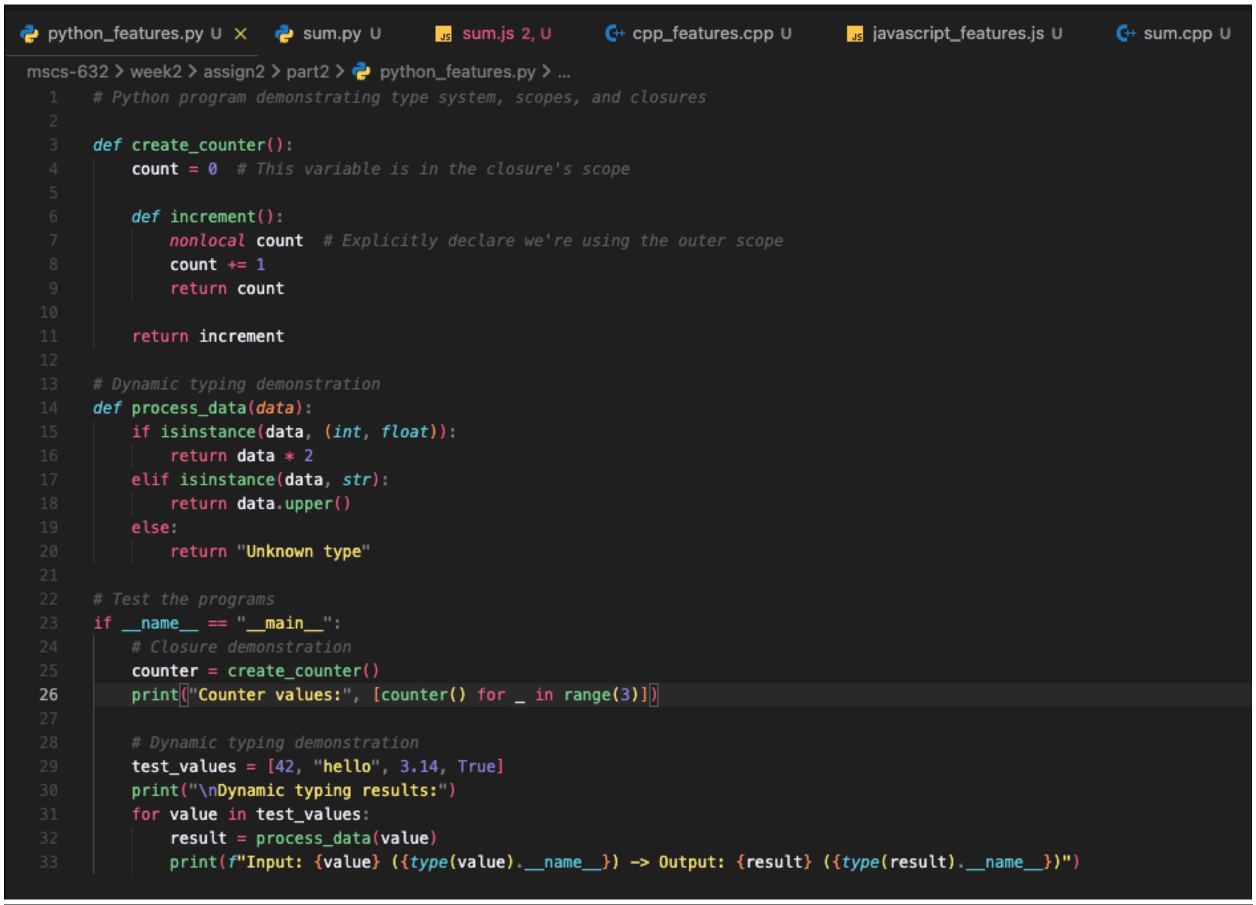
Differences in Error handling:

- Error detection: JavaScript and Python discover the error during the interpretation process while the C++ detection here happens during the compilation process.
- Message clarity: The error messages from all three demonstrations were perfectly clear as to what the syntax error was and where it originated from.
    - Python was concise as to what the error was and what caused it.
    - JavaScript added some unwanted details as to which parsing module threw the error (which may or may not be helpful).
    - C++ was concise but typically in my experience C++ compiler usually prints out the cascading error trace in the output, which might be cryptic and hard to make sense at times.

1.2 Closures

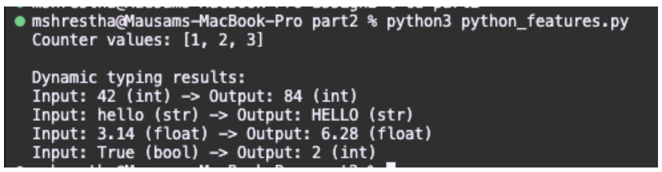
I have written programs in Python, JavaScript and C++ to capture how type systems and closures work in each language. Each sub-section in this section for each language, show the code we will execute and its output, and then we will have a sub-section discussing Type Systems, Scopes and Closures.

<u>Python</u>



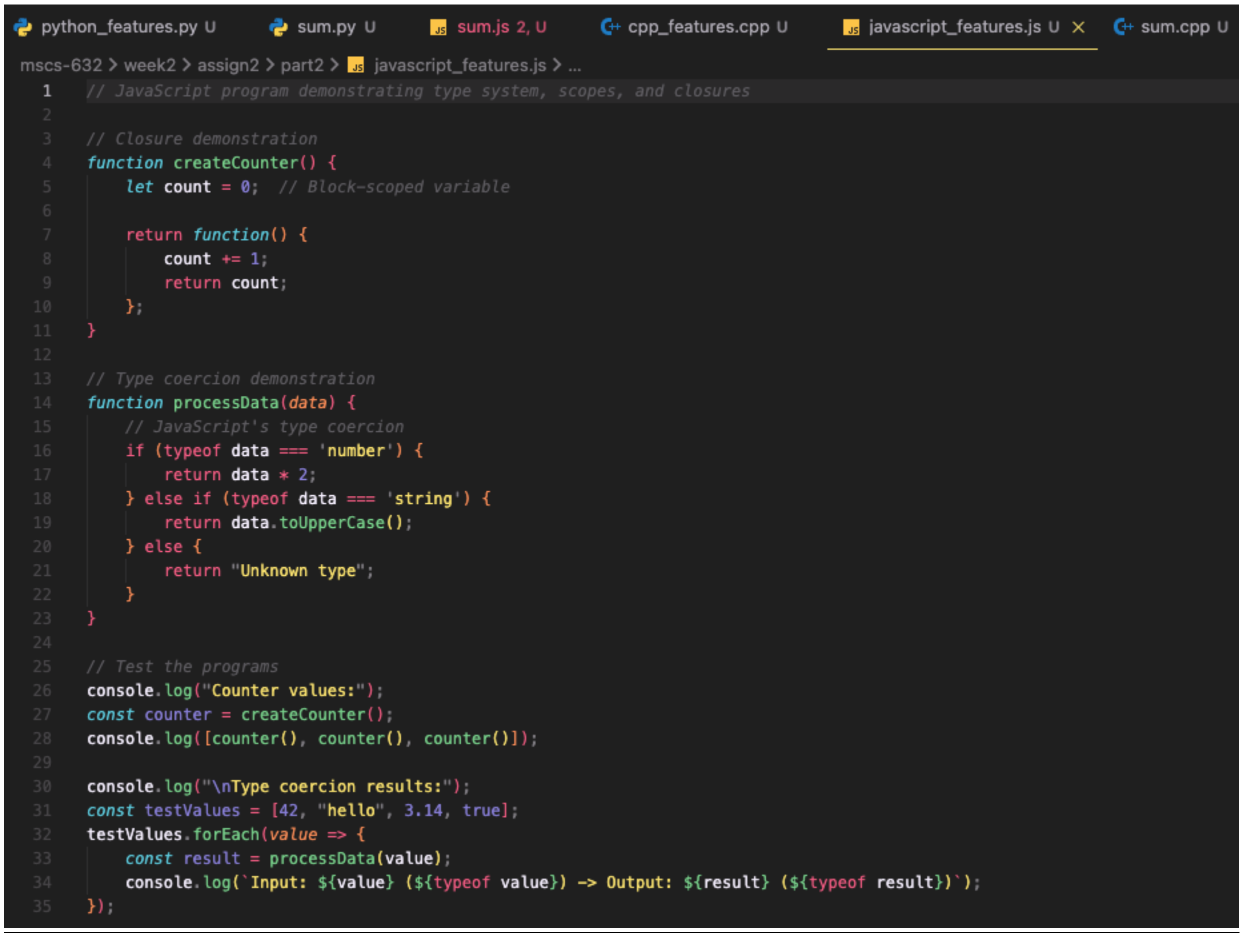
```python
# Python program demonstrating type system, scopes, and closures

def create_counter():
    count = 0  # This variable is in the closure's scope

    def increment():
        nonlocal count  # Explicitly declare we're using the outer scope
        count += 1
        return count

    return increment

# Dynamic typing demonstration
def process_data(data):
    if isinstance(data, (int, float)):
        return data * 2
    elif isinstance(data, str):
        return data.upper()
    else:
        return "Unknown type"

# Test the programs
if __name__ == "__main__":
    # Closure demonstration
    counter = create_counter()
    print("Counter values:", [counter() for _ in range(3)])

    # Dynamic typing demonstration
    test_values = [42, "hello", 3.14, True]
    print("\nDynamic typing results:")
    for value in test_values:
        result = process_data(value)
        print(f"Input: {value} ({type(value).__name__}) -> Output: {result} ({type(result).__name__})")
```

Output:



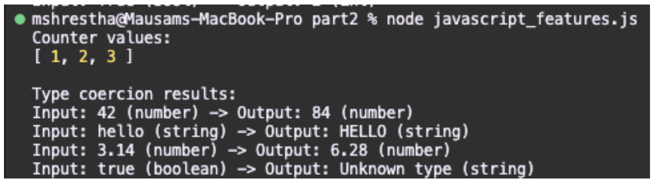
```
● mshrestha@Mausams-MacBook-Pro part2 % python3 python_features.py
  Counter values: [1, 2, 3]

  Dynamic typing results:
  Input: 42 (int) -> Output: 84 (int)
  Input: hello (str) -> Output: HELLO (str)
  Input: 3.14 (float) -> Output: 6.28 (float)
  Input: True (bool) -> Output: 2 (int)
```

JavaScript



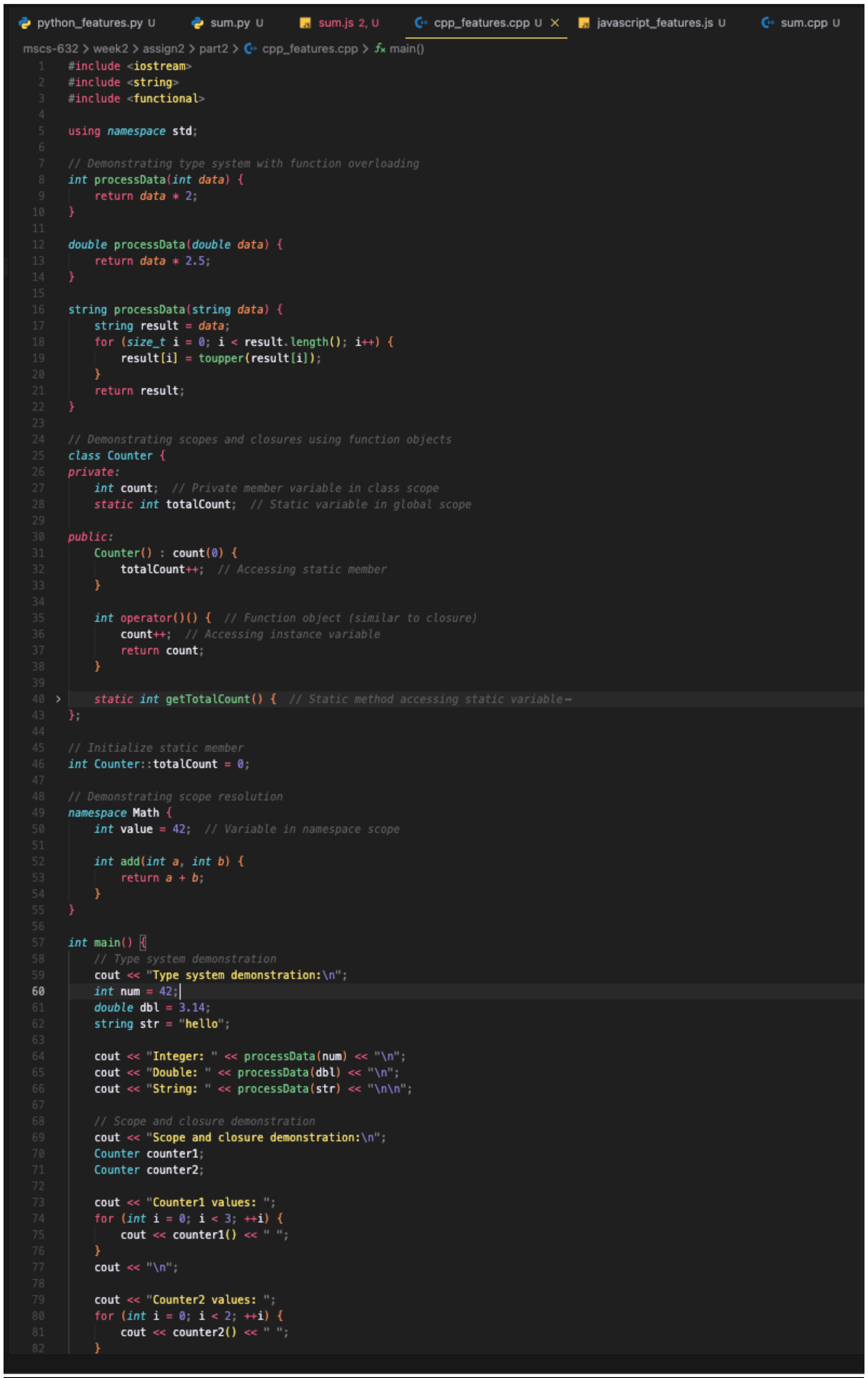
```javascript
// JavaScript program demonstrating type system, scopes, and closures

// Closure demonstration
function createCounter() {
    let count = 0;  // Block-scoped variable

    return function() {
        count += 1;
        return count;
    };
}

// Type coercion demonstration
function processData(data) {
    // JavaScript's type coercion
    if (typeof data === 'number') {
        return data * 2;
    } else if (typeof data === 'string') {
        return data.toUpperCase();
    } else {
        return "Unknown type";
    }
}

// Test the programs
console.log("Counter values:");
const counter = createCounter();
console.log([counter(), counter(), counter()]);

console.log("\nType coercion results:");
const testValues = [42, "hello", 3.14, true];
testValues.forEach(value => {
    const result = processData(value);
    console.log(`Input: ${value} (${typeof value}) -> Output: ${result} (${typeof result})`);
});
```

Output:



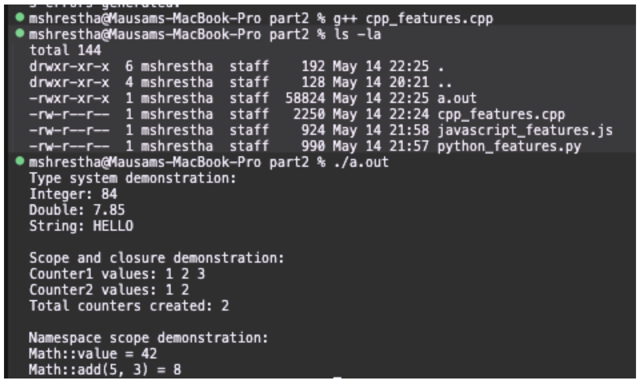
```
mshrestha@Mausams-MacBook-Pro part2 % node javascript_features.js
Counter values:
[ 1, 2, 3 ]

Type coercion results:
Input: 42 (number) -> Output: 84 (number)
Input: hello (string) -> Output: HELLO (string)
Input: 3.14 (number) -> Output: 6.28 (number)
Input: true (boolean) -> Output: Unknown type (string)
```

C++



```cpp
#include <iostream>
#include <string>
#include <functional>

using namespace std;

// Demonstrating type system with function overloading
int processData(int data) {
    return data * 2;
}

double processData(double data) {
    return data * 2.5;
}

string processData(string data) {
    string result = data;
    for (size_t i = 0; i < result.length(); i++) {
        result[i] = toupper(result[i]);
    }
    return result;
}

// Demonstrating scopes and closures using function objects
class Counter {
private:
    int count;  // Private member variable in class scope
    static int totalCount;  // Static variable in global scope

public:
    Counter() : count(0) {
        totalCount++;  // Accessing static member
    }

    int operator()() {  // Function object (similar to closure)
        count++;  // Accessing instance variable
        return count;
    }

    static int getTotalCount() {  // Static method accessing static variable⋯
};

// Initialize static member
int Counter::totalCount = 0;

// Demonstrating scope resolution
namespace Math {
    int value = 42;  // Variable in namespace scope

    int add(int a, int b) {
        return a + b;
    }
}

int main() {
    // Type system demonstration
    cout << "Type system demonstration:\n";
    int num = 42;
    double dbl = 3.14;
    string str = "hello";

    cout << "Integer: " << processData(num) << "\n";
    cout << "Double: " << processData(dbl) << "\n";
    cout << "String: " << processData(str) << "\n\n";

    // Scope and closure demonstration
    cout << "Scope and closure demonstration:\n";
    Counter counter1;
    Counter counter2;

    cout << "Counter1 values: ";
    for (int i = 0; i < 3; ++i) {
        cout << counter1() << " ";
    }
    cout << "\n";

    cout << "Counter2 values: ";
    for (int i = 0; i < 2; ++i) {
        cout << counter2() << " ";
    }
```

Output:



```
mshrestha@Mausams-MacBook-Pro part2 % g++ cpp_features.cpp
mshrestha@Mausams-MacBook-Pro part2 % ls -la
total 144
drwxr-xr-x  6 mshrestha  staff    192 May 14 22:25 .
drwxr-xr-x  4 mshrestha  staff    128 May 14 20:21 ..
-rwxr-xr-x  1 mshrestha  staff  58824 May 14 22:25 a.out
-rw-r--r--  1 mshrestha  staff   2250 May 14 22:24 cpp_features.cpp
-rw-r--r--  1 mshrestha  staff    924 May 14 21:58 javascript_features.js
-rw-r--r--  1 mshrestha  staff    990 May 14 21:57 python_features.py
mshrestha@Mausams-MacBook-Pro part2 % ./a.out
Type system demonstration:
Integer: 84
Double: 7.85
String: HELLO

Scope and closure demonstration:
Counter1 values: 1 2 3
Counter2 values: 1 2
Total counters created: 2

Namespace scope demonstration:
Math::value = 42
Math::add(5, 3) = 8
```

Type Systems:

- Python
  - Has dynamic typing resolved at runtime.
  - It has higher potential for runtime errors, since parsing does not help catch this.
  - Example: Look at the process_data() function, it takes in any input, and the function manually checks the type.
- JavaScript
  - Has dynamic typing resolved at runtime.
  - This has potential to cause unexpected behavior at runtime. (In JS, “2” + 2 = “22”), but might make some operations like generating a string by including a numerical output value easier.
  - Manual type checking needs to be enforced at runtime using the typeof operator for enforcing type strictness.

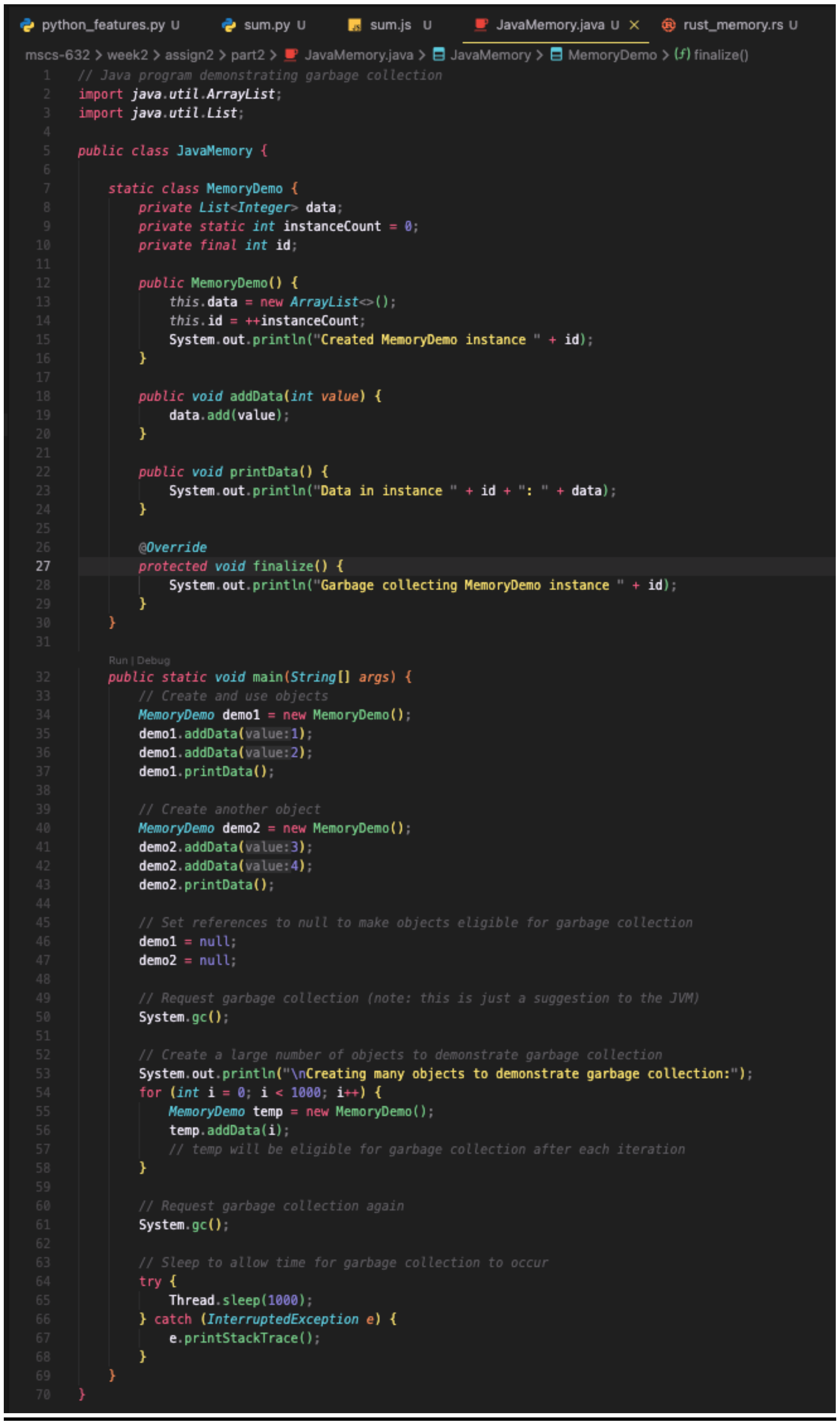
- C++
  - Has static typing that is resolved at compile time.
  - Types for variables and function returns needs to be explicitly declared.
  - Helps provide strict typing, but the amount of code increases.

Scopes and Closures:

- Python
  - We can use nonlocal keyword to modify variables out of scope. See, the function create_counter() for example.
  - Hierarchical scope structure (LEGB: local, enclosing, global, built-in)
- JavaScript
  - You can use let and const to declare variables that are block scoped.
  - The variables are captured by the closure.
- C++
  - C++ does not provide native support for closures.
  - We use objects to simulate similar functionality. See the class Counter for reference.

Part 2: Memory Management in Rust, Java and C++

Java



python_features.py U sum.py U sum.js U JavaMemory.java U × rust_memory.rs U

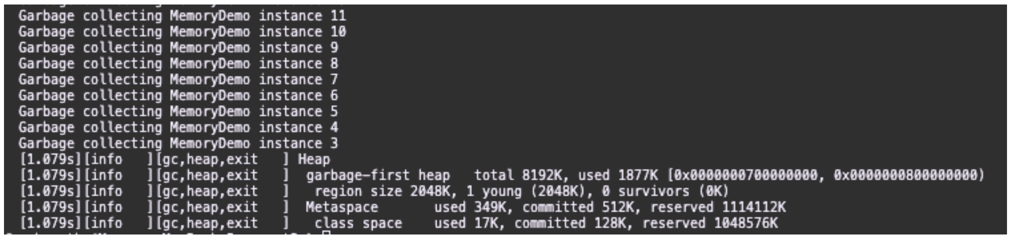
mscs-632 > week2 > assign2 > part2 > JavaMemory.java > JavaMemory > MemoryDemo > finalize()

```java
// Java program demonstrating garbage collection
import java.util.ArrayList;
import java.util.List;

public class JavaMemory {

    static class MemoryDemo {
        private List<Integer> data;
        private static int instanceCount = 0;
        private final int id;

        public MemoryDemo() {
            this.data = new ArrayList<>();
            this.id = ++instanceCount;
            System.out.println("Created MemoryDemo instance " + id);
        }

        public void addData(int value) {
            data.add(value);
        }

        public void printData() {
            System.out.println("Data in instance " + id + ": " + data);
        }

        @Override
        protected void finalize() {
            System.out.println("Garbage collecting MemoryDemo instance " + id);
        }
    }

    Run | Debug
    public static void main(String[] args) {
        // Create and use objects
        MemoryDemo demo1 = new MemoryDemo();
        demo1.addData(value:1);
        demo1.addData(value:2);
        demo1.printData();

        // Create another object
        MemoryDemo demo2 = new MemoryDemo();
        demo2.addData(value:3);
        demo2.addData(value:4);
        demo2.printData();

        // Set references to null to make objects eligible for garbage collection
        demo1 = null;
        demo2 = null;

        // Request garbage collection (note: this is just a suggestion to the JVM)
        System.gc();

        // Create a large number of objects to demonstrate garbage collection
        System.out.println("\nCreating many objects to demonstrate garbage collection:");
        for (int i = 0; i < 1000; i++) {
            MemoryDemo temp = new MemoryDemo();
            temp.addData(i);
            // temp will be eligible for garbage collection after each iteration
        }

        // Request garbage collection again
        System.gc();

        // Sleep to allow time for garbage collection to occur
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Java Output:

We compiled the code using javac JavaMemory.java and then ran with Garbage collection details enabled using:

- java -XX:+PrintGCDetails JavaMemory

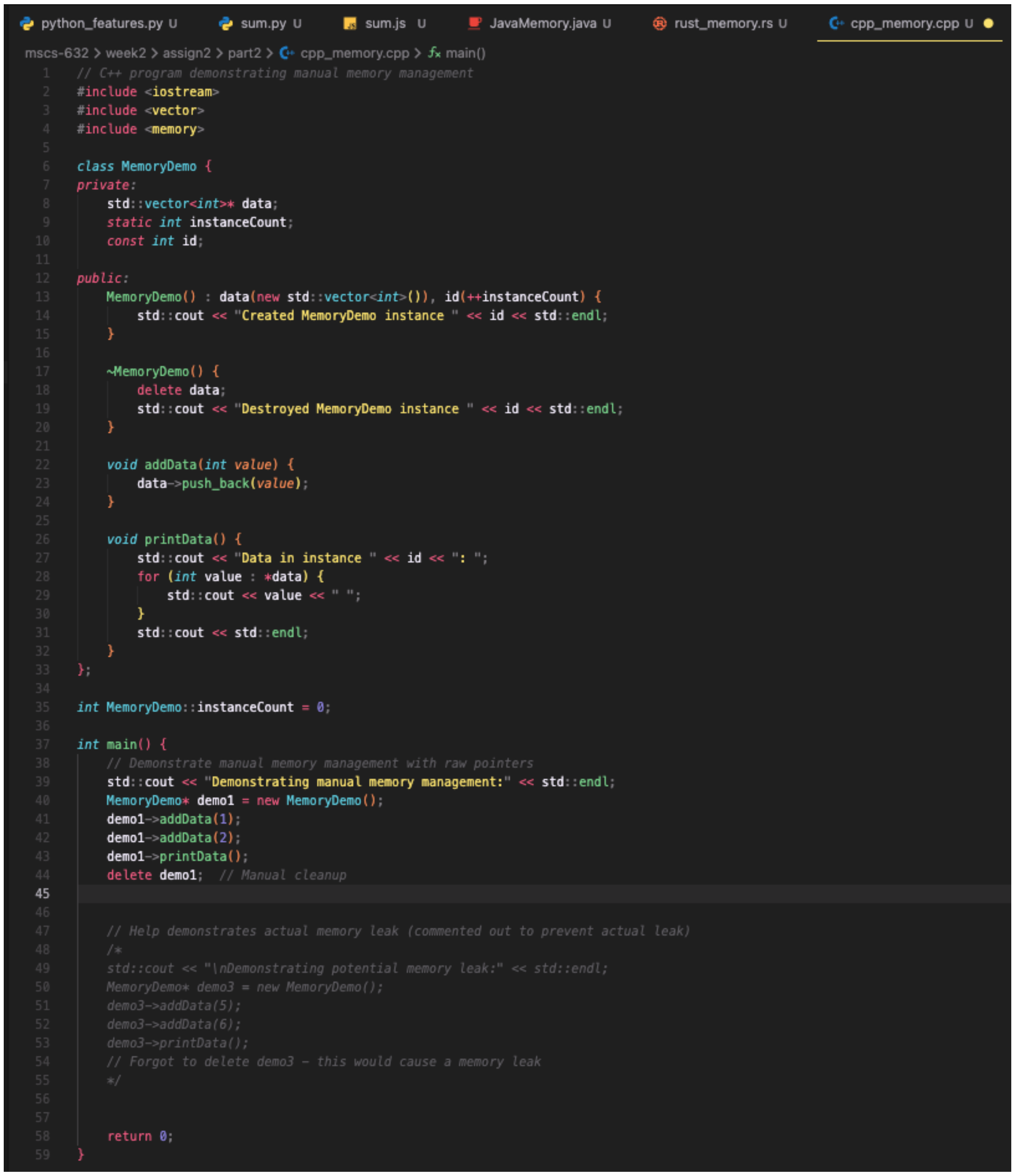


```
 Garbage collecting MemoryDemo instance 11
 Garbage collecting MemoryDemo instance 10
 Garbage collecting MemoryDemo instance 9
 Garbage collecting MemoryDemo instance 8
 Garbage collecting MemoryDemo instance 7
 Garbage collecting MemoryDemo instance 6
 Garbage collecting MemoryDemo instance 5
 Garbage collecting MemoryDemo instance 4
 Garbage collecting MemoryDemo instance 3
 [1.079s][info   ][gc,heap,exit   ] Heap
 [1.079s][info   ][gc,heap,exit   ]  garbage-first heap   total 8192K, used 1877K [0x0000000700000000, 0x0000000800000000)
 [1.079s][info   ][gc,heap,exit   ]   region size 2048K, 1 young (2048K), 0 survivors (0K)
 [1.079s][info   ][gc,heap,exit   ]  Metaspace       used 349K, committed 512K, reserved 1114112K
 [1.079s][info   ][gc,heap,exit   ]   class space    used 17K, committed 128K, reserved 1048576K
```

Explanation:

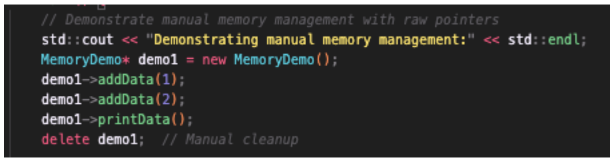
- If you look at the inner class MemoryDemo, we have overridden the finalize () method which is called by the Garbage Collector when it determines that the object does not have any references left in the application context.
- We are also using an ArrayList as a property in MemoryDemo, which helps us dynamically allocating memory for each Integer Object added to the list.
- Finally, in the main method we are creating MemoryDemo objects and then dereferencing the objects by setting the variable to null so that the actual objects would now become eligible for garbage collection since they don’t have any references.

C++:



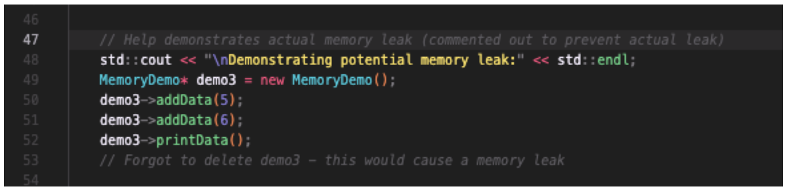
```cpp
// C++ program demonstrating manual memory management
#include <iostream>
#include <vector>
#include <memory>

class MemoryDemo {
private:
    std::vector<int>* data;
    static int instanceCount;
    const int id;

public:
    MemoryDemo() : data(new std::vector<int>()), id(++instanceCount) {
        std::cout << "Created MemoryDemo instance " << id << std::endl;
    }

    ~MemoryDemo() {
        delete data;
        std::cout << "Destroyed MemoryDemo instance " << id << std::endl;
    }

    void addData(int value) {
        data->push_back(value);
    }

    void printData() {
        std::cout << "Data in instance " << id << ": ";
        for (int value : *data) {
            std::cout << value << " ";
        }
        std::cout << std::endl;
    }
};

int MemoryDemo::instanceCount = 0;

int main() {
    // Demonstrate manual memory management with raw pointers
    std::cout << "Demonstrating manual memory management:" << std::endl;
    MemoryDemo* demo1 = new MemoryDemo();
    demo1->addData(1);
    demo1->addData(2);
    demo1->printData();
    delete demo1;  // Manual cleanup


    // Help demonstrates actual memory leak (commented out to prevent actual leak)
    /*
    std::cout << "\nDemonstrating potential memory leak:" << std::endl;
    MemoryDemo* demo3 = new MemoryDemo();
    demo3->addData(5);
    demo3->addData(6);
    demo3->printData();
    // Forgot to delete demo3 - this would cause a memory leak
    */


    return 0;
}
```

Manual Memory Management:



```cpp
// Demonstrate manual memory management with raw pointers
std::cout << "Demonstrating manual memory management:" << std::endl;
MemoryDemo* demo1 = new MemoryDemo();
demo1->addData(1);
demo1->addData(2);
demo1->printData();
delete demo1;  // Manual cleanup
```

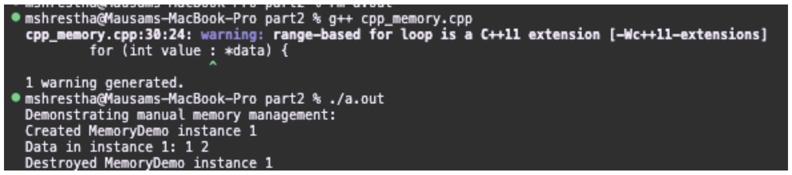
Creating a memory leak:



```
    // Help demonstrates actual memory leak (commented out to prevent actual leak)
    std::cout << "\nDemonstrating potential memory leak:" << std::endl;
    MemoryDemo* demo3 = new MemoryDemo();
    demo3->addData(5);
    demo3->addData(6);
    demo3->printData();
    // Forgot to delete demo3 - this would cause a memory leak
```

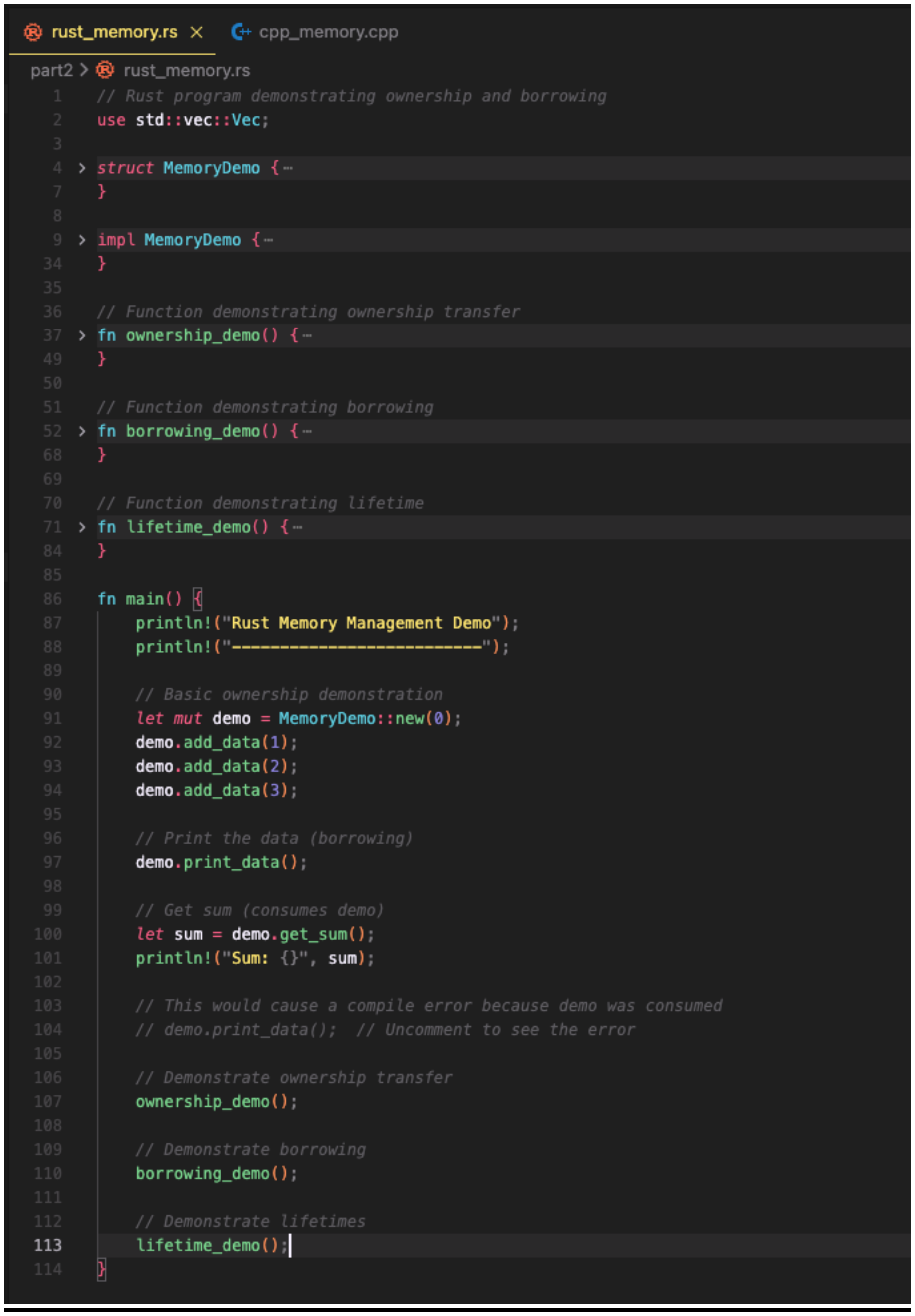
Output:

We are created a MemoryDemo class, and added in a destructor that prints out a message when the object gets explicitly deleted or goes out of scope. We then compiled the C++ file and ran the executable. As you can see, our destructor is getting called since we manually deleted the MemoryDemo object that we created.



```
● mshrestha@Mausams-MacBook-Pro part2 % g++ cpp_memory.cpp
  cpp_memory.cpp:30:24: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
          for (int value : *data) {
                         ^

  1 warning generated.
● mshrestha@Mausams-MacBook-Pro part2 % ./a.out
  Demonstrating manual memory management:
  Created MemoryDemo instance 1
  Data in instance 1: 1 2
  Destroyed MemoryDemo instance 1
```

Rust

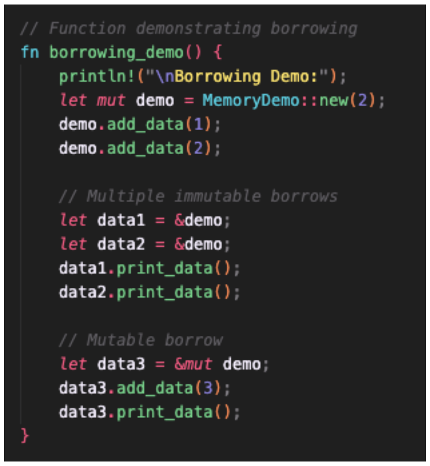


```rust
// Rust program demonstrating ownership and borrowing
use std::vec::Vec;

struct MemoryDemo {⋯
}

impl MemoryDemo {⋯
}

// Function demonstrating ownership transfer
fn ownership_demo() {⋯
}

// Function demonstrating borrowing
fn borrowing_demo() {⋯
}

// Function demonstrating lifetime
fn lifetime_demo() {⋯
}

fn main() {
    println!("Rust Memory Management Demo");
    println!("-------------------------");

    // Basic ownership demonstration
    let mut demo = MemoryDemo::new(0);
    demo.add_data(1);
    demo.add_data(2);
    demo.add_data(3);

    // Print the data (borrowing)
    demo.print_data();

    // Get sum (consumes demo)
    let sum = demo.get_sum();
    println!("Sum: {}", sum);

    // This would cause a compile error because demo was consumed
    // demo.print_data();  // Uncomment to see the error

    // Demonstrate ownership transfer
    ownership_demo();

    // Demonstrate borrowing
    borrowing_demo();

    // Demonstrate lifetimes
    lifetime_demo();
}
```

We will be looking at ownership and borrowing in Rust, the following snippets are examples of those concepts, and the output following at the very end helps connect the dots as to how its all coming together. We have compiled the rust code using:

- Rustc rust_memory.rs and then ran it using ./rust_memory

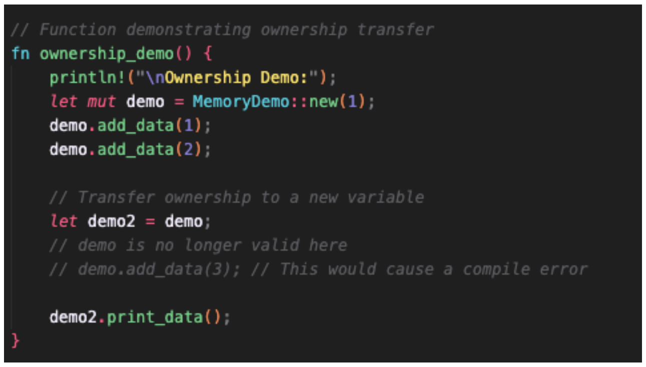
Borrowing:



```rust
// Function demonstrating borrowing
fn borrowing_demo() {
    println!("\nBorrowing Demo:");
    let mut demo = MemoryDemo::new(2);
    demo.add_data(1);
    demo.add_data(2);

    // Multiple immutable borrows
    let data1 = &demo;
    let data2 = &demo;
    data1.print_data();
    data2.print_data();

    // Mutable borrow
    let data3 = &mut demo;
    data3.add_data(3);
    data3.print_data();
}
```

In Rust, the concept of borrowing is the ability to have references to the data without owning it. There are two types of borrowing in swift, as demonstrated in the example above. For immutable borrows, the references don't have the ability to modify the data, they have read access to it. They can therefore print the contents, but cannot modify the underlying data. In, mutable borrows, only one borrow can exist at a time, and the borrowed underlying data can be modified.

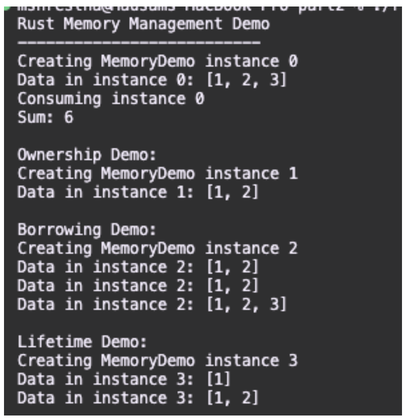
Ownership Transfer:



```rust
// Function demonstrating ownership transfer
fn ownership_demo() {
    println!("\nOwnership Demo:");
    let mut demo = MemoryDemo::new(1);
    demo.add_data(1);
    demo.add_data(2);

    // Transfer ownership to a new variable
    let demo2 = demo;
    // demo is no longer valid here
    // demo.add_data(3); // This would cause a compile error

    demo2.print_data();
}
```

In Rust, there can only be one owner for a value. We can transfer the ownership of a value from one variable to the other, but if we do that the previous owner becomes invalid. If incase the owner goes out of scope then the value is dropped and the memory is automatically freed.

Rust Program Output:



```
mshrestha@Mausams-MacBook-Pro part2 % ./r
Rust Memory Management Demo
---------------------------
Creating MemoryDemo instance 0
Data in instance 0: [1, 2, 3]
Consuming instance 0
Sum: 6

Ownership Demo:
Creating MemoryDemo instance 1
Data in instance 1: [1, 2]

Borrowing Demo:
Creating MemoryDemo instance 2
Data in instance 2: [1, 2]
Data in instance 2: [1, 2]
Data in instance 2: [1, 2, 3]

Lifetime Demo:
Creating MemoryDemo instance 3
Data in instance 3: [1]
Data in instance 3: [1, 2]
```