

Syntax, Semantics, and Memory Management

https://github.com/baralsamrat/MSCS632_Assignment_2

Samrat Baral

2025 Spring - Advanced Programming Languages (MSCS-632-B01) - Second Bi-term

University of the Cumberlands

Dr. Dax Bradley

March 16, 2025

Introduction

The assignment entailed an analysis of syntax, semantic disparities between various programming languages, and a comparison of memory management techniques.

Part 1 : Analyzing Syntax and Semantics

Syntax Errors Explained

The first part involved intentionally introducing syntax errors into code samples and then examining the resulting error messages produced by the interpreter or compiler of each language.

Language	Modification	Error Message
Python ▾	Replaced 0 with o ▾	NameError: name 'o' is not defined
Python ▾	Removed colon after function definition ▾	SyntaxError: invalid syntax
JavaScript ▾	Replaced 0 with o ▾	ReferenceError: o is not defined
JavaScript ▾	Introduced space in function call (e.g., calculate ... ▾	Syntax or parse error
C++ ▾	Replaced 0 with o ▾	error: 'o' was not declared in this scope
C++ ▾	Removed necessary semicolons ▾	Compiler error about missing semicolons

Key Difference

1. Type Checking:

- Python: Dynamic (runtime)
- JavaScript: Dynamic (runtime)
- C++: Static (compile time) (Tutorialspoint)

2. Scoping Rules:

- Python: Function-level (nonlocal for closures)
- JavaScript: Function and block-level
- C++: Block-level (Tutorialspoint)

3. Closure Implementation:

- Python: Implicit
- JavaScript: Implicit
- C++: Explicit (capture list in lambda expressions) (Tutorialspoint)

Part 2 : Memory Management

Memory Management in Rust, Java, and C++ Explained

This part compares how different languages handle dynamic memory allocation and deallocation.

Language	Approach	Error Handling
Rust	Uses ownership and borrowing, eliminating the need for a garbage collector.	Compile-time checks prevent issues like dangling pointers and memory leaks. (Tutorialspoint)
Java	Uses automatic garbage collection to reclaim unused memory.	Although memory leaks are less common, lingering references can still cause issues.
C++	Requires explicit allocation with new and deallocation with delete. (Tutorialspoint)	The programmer is responsible for preventing memory leaks and avoiding dangling pointers.

Screenshots

github.com /baralancet / MSCS632_Assignment_2 /src /1

```

1 def calculate_sum(arr): # Missing colon here
2     total = 0
3     for num in arr:
4         total += num
5     return total
6
7 numbers = [1, 2, 3, 4, 5]
8 result = calculate_sum(numbers)
9 print("Sum in Python:", result)
10
11 # Output
12 # Sum in Python: 15
13

```

Online Python Compiler

```

1 # Intentional error: using letter 'o' instead of digit 0 and e
2 missing colon in the function definition
3 def calculate_sun(arr): # Missing colon here
4     total = 0
5     for num in arr:
6         total +num
7     return total
8
9 numbers = [1, 2, 3, 4, 5]
10 result = calculate_sun(numbers)
11 print("Sum in Python-", result)
12

```

Sum in Python: 15

github.com / 2.py

```

1 # Python: Demonstrating dynamic typing and function scope (closures)
2 def outer():
3     x = 5
4     def inner():
5         # Python uses lexical scoping. 'nonlocal' allows modifying
6         nonlocal x
7         x += 1
8         return x
9     return inner
10
11 closure_function = outer()
12 print("Python closure call 1:", closure_function()) # 11
13 print("Python closure call 2:", closure_function()) # 12
14
15 # Output
16 # Python closure call 1: 11
17 # Python closure call 2: 12
18

```

Online Python Compiler

```

1 (closures)
2 - def outer():
3     x = 10
4 -     def inner():
5         # Python uses lexical scoping. 'nonlocal' allows
6         # modifying the variable in the enclosing scope.
7         nonlocal x
8         x += 1
9         return x
10    return inner
11
12 closure_function = outer()
13 print("Python closure call 1:", closure_function()) # 11
14 print("Python closure call 2:", closure_function()) # 12
15

```

Python closure call 1: 11
Python closure call 2: 12

github.com /froes /1.cpp

```

1 #include <iostream>
2 using namespace std;
3
4 int calculateSum(int arr[], int size) {
5     int total = 0;
6     for (int i = 0; i < size; i++) {
7         total += arr[i];
8     }
9     return total;
10 }
11
12 int main() {
13     int arr[] = {1, 2, 3, 4, 5}; // Missing semicolon at the end of the array declaration
14     int size = sizeof(numbers) / sizeof(numbers[0]);
15     int result = calculateSum(numbers, size);
16     cout << "Sum in C++: " << result << endl;
17     return 0;
18 }
19
20 # Output
21 # Sum in C++: 15
22

```

Online C++ Compiler

```

1 int total = 0;
2 for (int i = 0; i < size; i++) {
3     total += arr[i];
4 }
5 return total;
6
7 int main() {
8     int numbers[] = {1, 2, 3, 4, 5}; // Missing semicolon at the end of the array declaration
9     int size = sizeof(numbers) / sizeof(numbers[0]);
10    int result = calculateSum(numbers, size);
11    cout << "Sum in C++: " << result << endl;
12    return 0;
13 }
14

```

Sum in C++: 15

Assignment 2

6

The screenshot shows a browser window with two tabs open. The active tab is titled "Online C++ Compiler" and contains C++ code demonstrating the use of lambdas and mutable captures. The code defines a main function that prints two lambda calls, each capturing 'x' by value and marking it as mutable. The output shows the results of these two lambda executions.

```
// C++ Demystifying static typing and lambda closures
#include <iostream>
#include <functional>
using namespace std;

int main() {
    int x = 10;
    // C++ lambda capturing 'x' by value and marked mutable to allow modification.
    auto inner = [x](int add) mutable -> int {
        x += add;
        return x;
    };
    cout << "C++ lambda call 1: " << inner(1) << endl; // 11
    cout << "C++ lambda call 2: " << inner(1) << endl; // 12 (x is updated within the lambda)
    return 0;
}

// Output
// C++ lambda call 1: 11
// C++ lambda call 2: 12
```

github.com

MSCS632_Assignment_2 / src / 3.cpp

barasmatr Update 3.cpp 0d8c74c · now

18 lines (16 loc) · 446 Bytes

Code	Blame
1 // C++: Manual memory management with new and delete 2 #include <iostream> 3 using namespace std; 4 5 v int main() { 6 // Dynamically allocate an array of 5 integers 7 int* numbers = new int[5]; 8 for (int i = 0; i < 5; i++) { 9 numbers[i] = i; 10 } 11 cout << "Numbers allocated in C++." << endl; 12 // Manually free the allocated memory to prevent memory leak 13 delete[] numbers; 14 return 0; 15 } 16 17 // Output 18 // Numbers allocated in C++	Raw

tutorialspoint.com

Cengage Naruto https://...MaltonCY CodePath MyMav

Execute Source Code Share Help

```
3 using namespace std;  
4  
5 int main() {  
6     // Dynamically allocate an array of 5 integers  
7     int* numbers = new int[5];  
8     for (int i = 0; i < 5; i++) {  
9         numbers[i] = i;  
10    }  
11    cout << "Numbers allocated in C++." << endl;  
12    // Manually free the allocated memory to prevent memory  
13    // leaks  
14    delete[] numbers;  
15 }
```

Numbers allocated in C++.

Assignment 2

7

10:03AM Sun Mar 16

github.com

baralsamrat / **MSCS632_Assignment_2**

Code Issues Pull requests Actions Projects

Files main

MSCS632_Assignment_2/src/1/1.js

baralsamrat Update 1.js ddb29ed · now

14 lines (12 loc) · 215 Bytes

Code Blame Raw ⌂ ⌂ ⌂

```
1 function calculateSum(arr) {
2     let total = 0;
3     for (let num of arr) {
4         total += num;
5     }
6     return total;
7 }
8
9 let numbers = [1, 2, 3, 4, 5];
10 let result = calculateSum(numbers);
11 console.log("Sum in JavaScript:", result);
12
13 // Output:
14 // Sum in JavaScript: 15
```

Web View

JavaScript Playgro... Share Learn Sign In

index.html x script.js x

```
1 function calculateSum(arr) {
2     let total = 0;
3     for (let num of arr) {
4         total += num;
5     }
6     return total;
7 }
8
9 let numbers = [1, 2, 3, 4, 5];
10 // Intentional error: space in function name call, should t
11 let result = calculateSum(numbers);
12 console.log('Sum in JavaScript:', result);
13
```

Console x

Sum in JavaScript: 15

Web View

10:04AM Sun Mar 16

github.com

baralsamrat / **MSCS632_Assignment_2/src/2/2.js**

baralsamrat Update 2.js 4281c8c · now

19 lines (16 loc) · 515 Bytes

Code Blame Raw ⌂ ⌂ ⌂

```
1 // JavaScript: Demonstrating lexical scoping and closures
2 function outer() {
3     let x = 10;
4     function inner() {
5         // JavaScript functions create closures capturing variables from their lexical environment
6         x += 1;
7         return x;
8     }
9     return inner;
10 }
11
12 const closureFunction = outer();
13 console.log("JavaScript closure call 1:", closureFunction());
14 console.log("JavaScript closure call 2:", closureFunction());
15
16
17 // Output
18 // JavaScript closure call 1: 11
19 // JavaScript closure call 2: 12
```

Web View

JavaScript Playgro... Share Learn Sign In

index.html x script.js x

```
1 (function () {
2     "use strict";
3     var outer = (function () {
4         var x = 10;
5         return (function () {
6             "use strict";
7             var inner = function () {
8                 "use strict";
9                 var x = outer.x + 1;
10                return x;
11            };
12            return inner;
13        }();
14        return outer;
15    }());
16    var closureFunction = outer();
17    console.log("JavaScript closure call 1:", closureFunction());
18    console.log("JavaScript closure call 2:", closureFunction());
19 })();
```

Console x

JavaScript closure call 1: 11
JavaScript closure call 2: 12

Web View

10:31AM Sun Mar 16

github.com

Edit Preview Spaces 2 No wrap

```
1 import os
2 import psutil
3 import time
4
5 def print_memory_usage(note):
6     process = psutil.Process(os.getpid())
7     mem = process.memory_info().rss / (1024 * 1024) # Convert bytes to MB
8     print(f"[{note}] Memory usage: ({mem:.2f} MB")
9
10 print_memory_usage("Start")
11
12 # Allocate a large list
13 arr = [i for i in range(1000000)]
14 print_memory_usage("After allocation")
15
16 # Pause to observe memory before deletion
17 time.sleep(1)
18
19 # Delete the list and force garbage collection
20 del arr
21 time.sleep(1)
22 print_memory_usage("After deletion")
23
24 # Output
25 # [Start] Memory usage: 103.16 MB
26 # [After allocation] Memory usage: 141.42 MB
27 # [After deletion] Memory usage: 105.19 MB
```

Use Control + Shift + n to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.

File Edit View Insert Run Help Share RAM Disk

Untitled3.ipynb

```
[1]: pip install psutil
Requirement already satisfied: psutil in /usr/local/lib/python3.6/dist-packages
```

In [2]:

```
import os
import psutil
import time

def print_memory_usage(note):
    process = psutil.Process(os.getpid())
    mem = process.memory_info().rss / (1024 * 1024) # Convert bytes to MB
    print(f"[{note}] Memory usage: ({mem:.2f} MB")

print_memory_usage("Start")
```

Allocate a large list

```
arr = [i for i in range(1000000)]
print_memory_usage("After allocation")
```

Pause to observe memory before deletion

```
time.sleep(1)
```

Delete the list and force garbage collection

```
del arr
```

Copy Find Selection Look Up Translate Search Web

10:31AM Sun Mar 16 ***

github.com

Edit Preview Spaces 2 No wrap

```

1 import os
2 import psutil
3 import time
4
5 def print_memory_usage(note):
6     process = psutil.Process(os.getpid())
7     mem = process.memory_info().rss / (1024 * 1024) # Convert bytes to MB
8     print(f"[{note}] Memory usage: ({mem:.2f} MB")
9
10 print_memory_usage("Start")
11
12 # Allocate a large list
13 arr = [i for i in range(1000000)]
14 print_memory_usage("After allocation")
15
16 # Pause to observe memory before deletion
17 time.sleep(1)
18
19 # Delete the list and force garbage collection
20 del arr
21 time.sleep(1)
22 print_memory_usage("After deletion")
23
24 # Output
25 # [Start] Memory usage: 103.16 MB
26 # [After allocation] Memory usage: 141.42 MB
27 # [After deletion] Memory usage: 105.19 MB
28

```

Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.

File Edit View Insert Run File Share IDE, Edit Compile https://co...-MafcnCY CodePath MyMav ...

[1] pip install psutil
Requirement already satisfied: psutil in /usr/local/lib/python3.8/dist-packages

[x] import os
import psutil
import time

def print_memory_usage(note):
 process = psutil.Process(os.getpid())
 mem = process.memory_info().rss / (1024 * 1024) # Convert bytes to MB
 print(f"[{note}] Memory usage: ({mem:.2f} MB")

print_memory_usage("Start")

Allocate a large list
arr = [i for i in range(1000000)]
print_memory_usage("After allocation")
|
Pause to observe memory before deletion
time.sleep(1)

Delete the list and force garbage collection
del arr
time.sleep(1)
print_memory_usage("After deletion")

[Start] Memory usage: 103.16 MB
[After allocation] Memory usage: 141.42 MB
[After deletion] Memory usage: 105.19 MB ✓ 2s completed at 10:30 AM

10:34 AM Sun Mar 16 ***

github.com

Edit Preview Spaces 2 No wrap

```

2 #include <sys/resource.h>
3 #include <unistd.h>
4 #include <vector>
5 #include <string>
6
7 using namespace std;
8
9 void printMemoryUsage(const string &note) {
10     struct rusage usage;
11     getrusage(RUSAGE_SELF, &usage);
12     // ru_maxrss is in kilobytes on Linux
13     cout << "After allocation" << note << " Memory usage: " << usage.ru_maxrss / 1024.0;
14 }
15
16 int main() {
17     printMemoryUsage("Start");
18
19     // Allocate a vector with 1,000,000 integers
20     vector<int> v;
21     for (int i = 0; i < 1000000; i++) {
22         v.push_back(i);
23     }
24     printMemoryUsage("After allocation");
25
26     // Clear the vector to free memory
27     v.clear();
28     printMemoryUsage("After deletion");
29
30     return 0;
31 }

```

Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.

File Execute Source Code Share Help

[Start] Memory usage: 2.83906 MB
[After allocation] Memory usage: 7.59375 MB
[After deletion] Memory usage: 7.59375 MB

10:43 AM Sun Mar 16 ***

github.com

Edit Preview Spaces 2 No wrap

```

1 use sysinfo::(System, SystemExt);
2
3 fn print_memory_usage(note: &str) {
4     let mut sys = System::new_all();
5     sys.refresh_all();
6     let used_memory = sys.used_memory();
7     println!("{} Memory usage: {:.2}", note, used_memory);
8 }
9
10 fn main() {
11     print_memory_usage("Start");
12
13     // Allocate a vector with 1,000,000 integers
14     let v: Vec<i32> = (0..1_000_000).collect();
15     print_memory_usage("After allocation");
16
17     // Drop the vector and observe memory usage
18     drop(v);
19     print_memory_usage("After deletion");
20 }

```

File Online Java Untitled3.ipynb play.rust-lang.org ...

[2] fn print_memory_usage(note: &str) {
3 println!("{} Memory usage info unavailable in this environment", note);
4 }

fn main() {
 print_memory_usage("Start");
 // Allocate a vector with 1,000,000 integers
10 let v: Vec<i32> = (0..1_000_000).collect();
11 print_memory_usage("After allocation");
12
 // Drop the vector and observe memory usage
14 drop(v);
15 print_memory_usage("After deletion");
16 }

Execution Format Clippy Miri Macro expansion Share Close Standard Error
Compiling playground v0.0.1 (/playground)
Finished 'dev' profile [unoptimized + debuginfo] target(s) in 1.23s
Running 'target/debug/playground'

Standard Output
[Start] (Memory usage info unavailable in this environment)
[After allocation] (Memory usage info unavailable in this environment)
[After deletion] (Memory usage info unavailable in this environment)

Conclusion

Memory management and profiling also differ across the languages. The key semantic differences between Python, JavaScript, and C++ are evident in their type checking, memory management, and closure implementation. (Tutorialspoint)

- Python utilizes automatic memory management (garbage collection) and provides memory usage details via the psutil module. Memory is allocated for large data structures and released once objects are deleted. (Tutorialspoint)
- JavaScript also employs garbage collection. Memory usage is monitored with Node.js's process.memoryUsage(). Memory snapshots reveal that memory usage decreases once objects are dereferenced. (Tutorialspoint)
- Python and JavaScript use dynamic typing, which determines variable types at runtime. This facilitates rapid development but can introduce runtime type errors.
- Python and JavaScript depend on garbage collection for memory management. This simplifies development but can lead to unpredictable pauses.
- Closures are implicit in Python and JavaScript, streamlining development. (Tutorialspoint)
- C++ uses manual memory management, affording precise control but increasing the risk of memory leaks. (Tutorialspoint)
- C++ mandates explicit lambda capture lists, which can be more verbose but offer greater clarity and control.
- C++ employs static typing, determining types at compile time, which enhances reliability and performance but necessitates more explicit type declarations.
- C++ depends on manual allocation/deallocation (or smart pointers). POSIX's getrusage

function (or tools like Valgrind) are used for memory profiling. Improper memory handling can result in memory leaks, detectable with advanced profiling tools.

(Tutorialspoint)

References

Online programming compilers. Tutorialspoint. (n.d.).

<https://www.tutorialspoint.com/online-programming-compilers.html>