Analyzing Syntax and Semantics

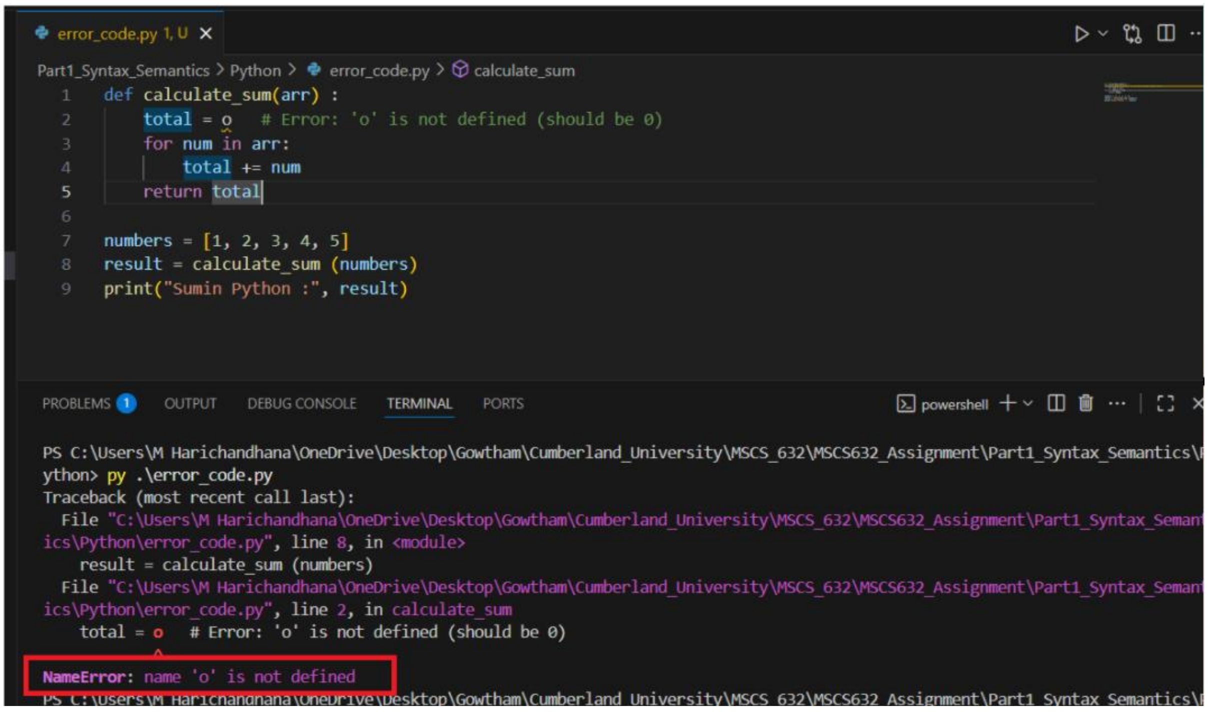
Submitted by

Shiva Gowtham Kumar Vidiyala

Instructor Name : Jay Thom

Advanced Programming Language

Python Error Explanation



```python
def calculate_sum(arr) :
    total = o    # Error: 'o' is not defined (should be 0)
    for num in arr:
        total += num
    return total

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

```
PS C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semantics\P
ython> py .\error_code.py
Traceback (most recent call last):
  File "C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semant
ics\Python\error_code.py", line 8, in <module>
    result = calculate_sum (numbers)
  File "C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semant
ics\Python\error_code.py", line 2, in calculate_sum
    total = o   # Error: 'o' is not defined (should be 0)
            ^
NameError: name 'o' is not defined
PS C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semantics\P
```

The error message is usually clear and includes the line number, file, and type of error (NameError, SyntaxError)

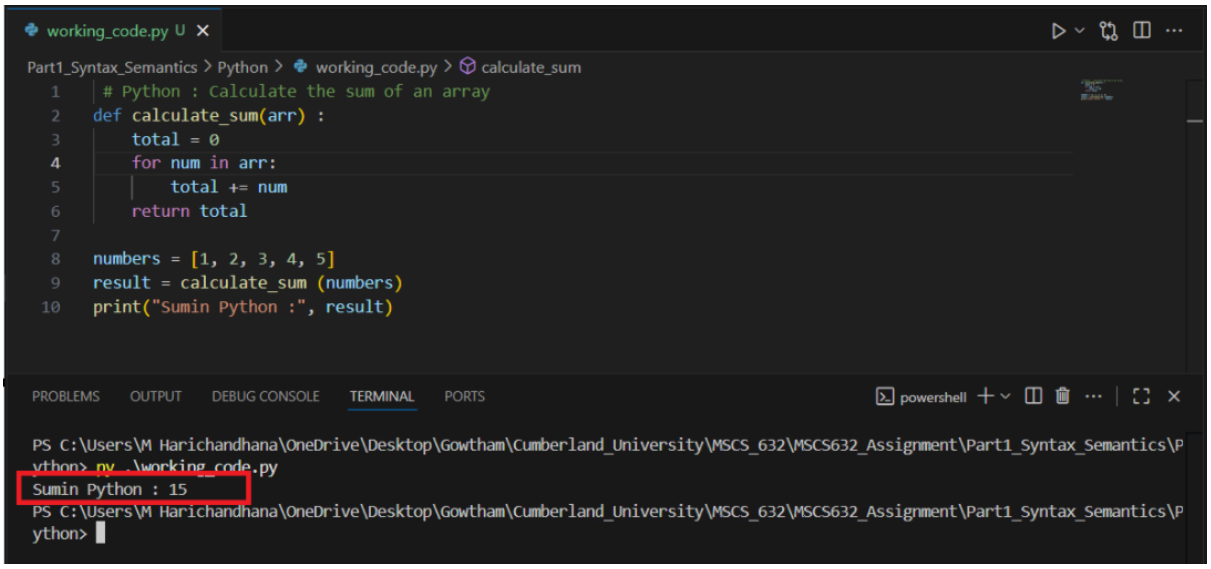
total = o — Here, o is a typo. It should be the number 0.

Python raises a **NameError**, as it treats o as an undefined variable.

Python is **interpreted**, so the error is caught at **runtime**, not at compilation.

After changing o to 0, we got the correct result.

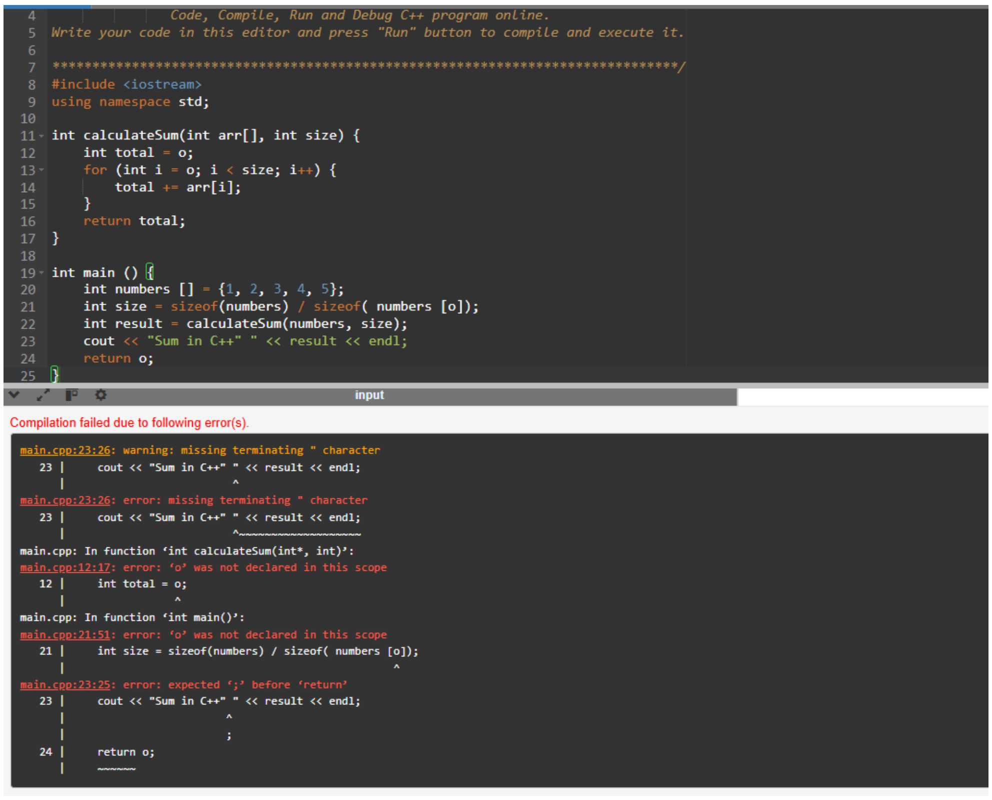
Here the python interpreter parses the whole file first and stops at the first syntax error, showing the file/line/column and a caret ^ pointing near the offending token.


working_code.py U ×
Part1_Syntax_Semantics > Python > working_code.py > calculate_sum
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell

```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("Sumin Python :", result)
```

```
PS C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semantics\P
ython> py .\working_code.py
Sumin Python : 15
PS C:\Users\M Harichandhana\OneDrive\Desktop\Gowtham\Cumberland_University\MSCS_632\MSCS632_Assignment\Part1_Syntax_Semantics\P
ython> 
```

C++ Error Explanation



```cpp
4                     Code, Compile, Run and Debug C++ program online.
5  Write your code in this editor and press "Run" button to compile and execute it.
6
7  *******************************************************************************/
8  #include <iostream>
9  using namespace std;
10
11 int calculateSum(int arr[], int size) {
12     int total = o;
13     for (int i = o; i < size; i++) {
14         total += arr[i];
15     }
16     return total;
17 }
18
19 int main () {
20     int numbers [] = {1, 2, 3, 4, 5};
21     int size = sizeof(numbers) / sizeof( numbers [o]);
22     int result = calculateSum(numbers, size);
23     cout << "Sum in C++" " << result << endl;
24     return o;
25 }
```

Compilation failed due to following error(s).

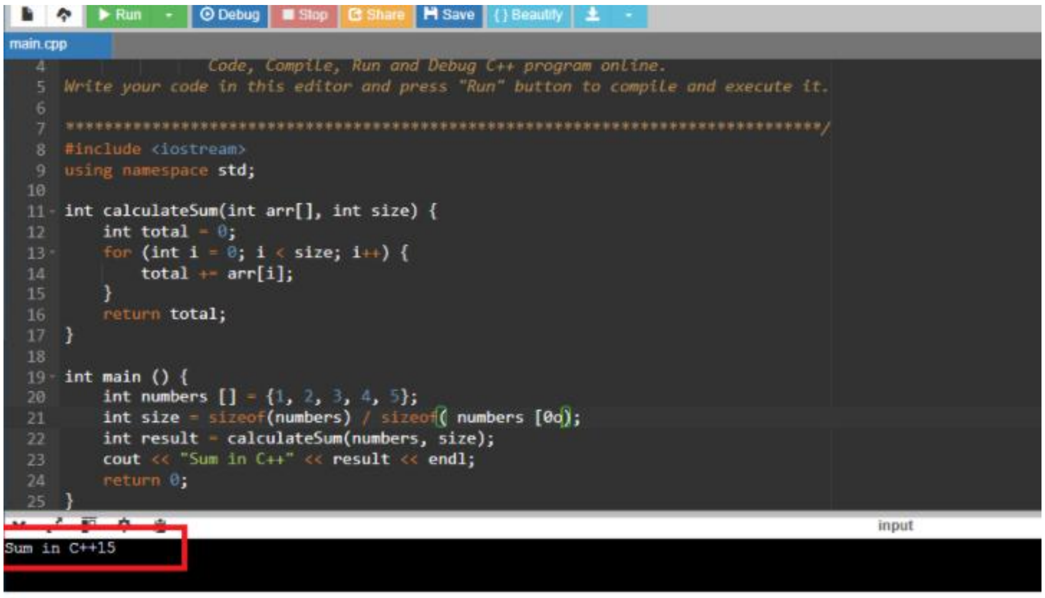
```
main.cpp:23:26: warning: missing terminating " character
   23 |     cout << "Sum in C++" " << result << endl;
      |                          ^
main.cpp:23:26: error: missing terminating " character
   23 |     cout << "Sum in C++" " << result << endl;
      |                          ^~~~~~~~~~~~~~~~~~~~
main.cpp: In function ‘int calculateSum(int*, int)’:
main.cpp:12:17: error: ‘o’ was not declared in this scope
   12 |     int total = o;
      |                 ^
main.cpp: In function ‘int main()’:
main.cpp:21:51: error: ‘o’ was not declared in this scope
   21 |     int size = sizeof(numbers) / sizeof( numbers [o]);
      |                                                   ^
main.cpp:23:25: error: expected ‘;’ before ‘return’
   23 |     cout << "Sum in C++" " << result << endl;
      |                         ^
      |                         ;
   24 |     return o;
      |     ~~~~~~
```

o is undefined in all places; it should be 0.

cout << "Sum in C++" " << result << endl; — invalid string concatenation.

return o; — again undefined.

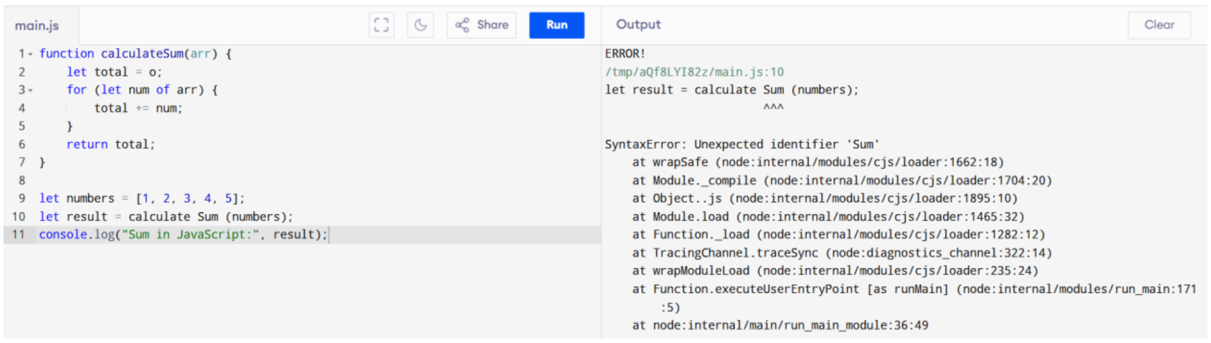
C++ handles it: A separate compiler parses and type-checks first. It may produce multiple cascading diagnostics, but the first error is usually the true cause (missing ;).



```cpp
                Code, Compile, Run and Debug C++ program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/
#include <iostream>
using namespace std;

int calculateSum(int arr[], int size) {
    int total = 0;
    for (int i = 0; i < size; i++) {
        total += arr[i];
    }
    return total;
}

int main () {
    int numbers [] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof( numbers [0o);
    int result = calculateSum(numbers, size);
    cout << "Sum in C++" << result << endl;
    return 0;
}
```

```
Sum in C++15
```

JavaScript Error Explanation



```javascript
function calculateSum(arr) {
    let total = o;
    for (let num of arr) {
        total += num;
    }
    return total;
}

let numbers = [1, 2, 3, 4, 5];
let result = calculate Sum (numbers);
console.log("Sum in JavaScript:", result);
```

```
ERROR!
/tmp/aQf8LYI82z/main.js:10
let result = calculate Sum (numbers);
                       ^^^

SyntaxError: Unexpected identifier 'Sum'
    at wrapSafe (node:internal/modules/cjs/loader:1662:18)
    at Module._compile (node:internal/modules/cjs/loader:1704:20)
    at Object..js (node:internal/modules/cjs/loader:1895:10)
    at Module.load (node:internal/modules/cjs/loader:1465:32)
    at Function._load (node:internal/modules/cjs/loader:1282:12)
    at TracingChannel.traceSync (node:diagnostics_channel:322:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:235:24)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:171
        :5)
    at node:internal/main/run_main_module:36:49
```

The error happens due to **syntax error**

Here JavaScript treats calculate and Sum as two separate identifiers because of the space

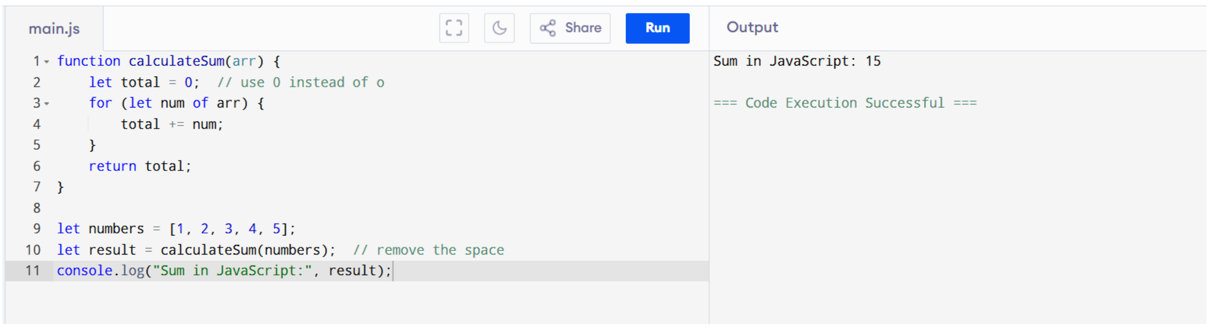
Function names cannot have spaces, so the interpreter throws:

SyntaxError: Unexpected identifier 'Sum'

There is one more error: o is not defined. It should be the number 0 (zero), not the letter o.

Here **JavaScript** engine parses the script before execution and fails fast on malformed structure (e.g., unmatched braces), usually reporting the first unexpected token.

Working Code



```javascript
function calculateSum(arr) {
    let total = 0;  // use 0 instead of o
    for (let num of arr) {
        total += num;
    }
    return total;
}

let numbers = [1, 2, 3, 4, 5];
let result = calculateSum(numbers);  // remove the space
console.log("Sum in JavaScript:", result);
```

```
Sum in JavaScript: 15

=== Code Execution Successful ===
```

Quick comparison: syntax-error handling

Python: stops at first syntax error during parsing; message is short and points to the exact spot (often “expected …”).

JavaScript: also stops at first parsing error; messages often say “Unexpected token …” and highlight the first structural inconsistency.

C++: compiler may emit several diagnostics due to cascading errors after the first real mistake; messages are detailed (sometimes verbose) and can include notes and hints.

Type System

Python: Dynamic → easy and flexible, but type errors show up only when running; slower because compiler knows less.

JavaScript: Dynamic at runtime, but **TypeScript** adds optional compile-time checks → safer and better tooling without changing runtime.

C++: Static → errors caught early, highly optimized code, but more verbose and complex.

Why it matters: Affects development speed, safety, and performance.

Closures & Scoping

Python: Captures names (late binding) → loop gotchas unless fixed with defaults.

JavaScript: Closures are everywhere; let/const give safer block scope than old var.

C++: Lambdas with explicit capture lists (by value or ref) → powerful but must manage correctness and performance.

Memory Management

Python & JS: Garbage collection → no manual cleanup, easier to code, but occasional pauses.

C++: Manual control (stack, RAII, smart pointers) → predictable and efficient, but more work for developer.

Conclusion: Python favours speed of writing, JavaScript balances flexibility with optional safety, and C++ emphasizes control and performance but requires discipline.

GITHUB LINK:

https://github.com/gowthamvidi/MSCS632_Assignment.git