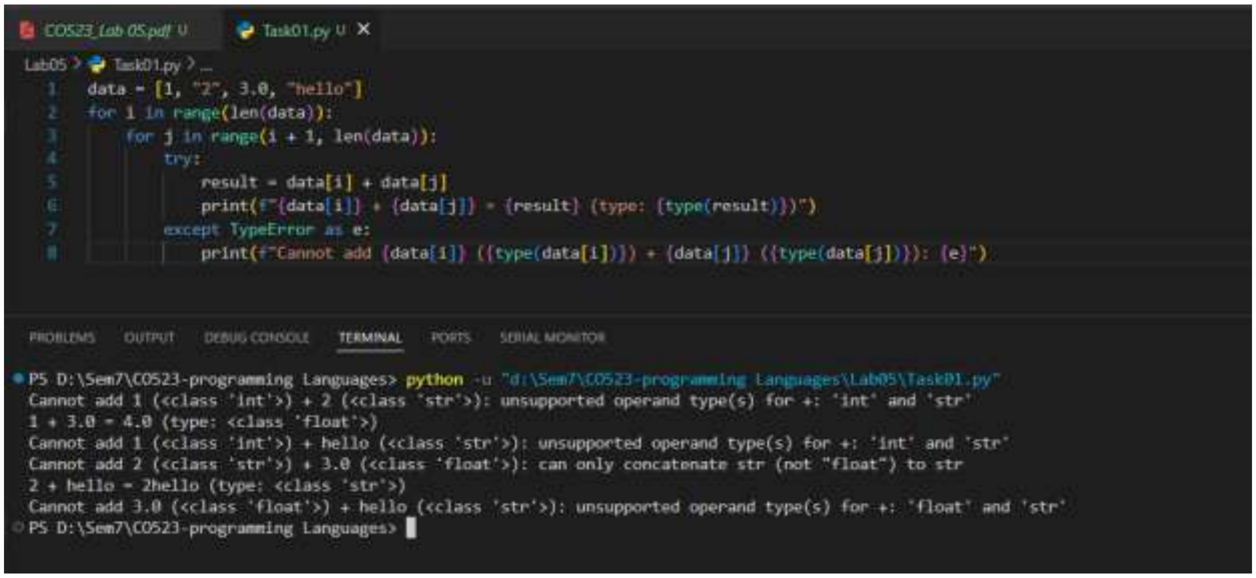
Department of Computer Engineering University of Peradeniya
CO523 - Programming Languages
Lab 05: Data Types, Type Systems, and Type Checking

E/20/449
WIJEWARDHANA A.R.S.S.

Task 01



```python
data = [1, "2", 3.0, "hello"]
for i in range(len(data)):
    for j in range(i + 1, len(data)):
        try:
            result = data[i] + data[j]
            print(f"{data[i]} + {data[j]} = {result} (type: {type(result)})")
        except TypeError as e:
            print(f"Cannot add {data[i]} ({type(data[i])}) + {data[j]} ({type(data[j])}): {e}")
```

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\Lab05\Task01.py"
Cannot add 1 (<class 'int'>) + 2 (<class 'str'>): unsupported operand type(s) for +: 'int' and 'str'
1 + 3.0 = 4.0 (type: <class 'float'>)
Cannot add 1 (<class 'int'>) + hello (<class 'str'>): unsupported operand type(s) for +: 'int' and 'str'
Cannot add 2 (<class 'str'>) + 3.0 (<class 'float'>): can only concatenate str (not "float") to str
2 + hello = 2hello (type: <class 'str'>)
Cannot add 3.0 (<class 'float'>) + hello (<class 'str'>): unsupported operand type(s) for +: 'float' and 'str'
PS D:\Sem7\CO523-programming Languages>
```

Observed Output
When you run this script, you will see output similar to this:

1. Cannot add 1 (<class 'int'>) + 2 (<class 'str'>): unsupported operand type(s) for +: 'int' and 'str'
2. 1 + 3.0 = 4.0 (type: <class 'float'>)
3. Cannot add 1 (<class 'int'>) + hello (<class 'str'>): unsupported operand type(s) for +: 'int' and 'str'
4. Cannot add 2 (<class 'str'>) + 3.0 (<class 'float'>): can only concatenate str (not "float") to str
5. 2 + hello = 2hello (type: <class 'str'>)
6. Cannot add 3.0 (<class 'float'>) + hello (<class 'str'>): unsupported operand type(s) for +: 'float' and 'str'

Explanation and Classification
Based on these results :

A. Python is Dynamically Typed

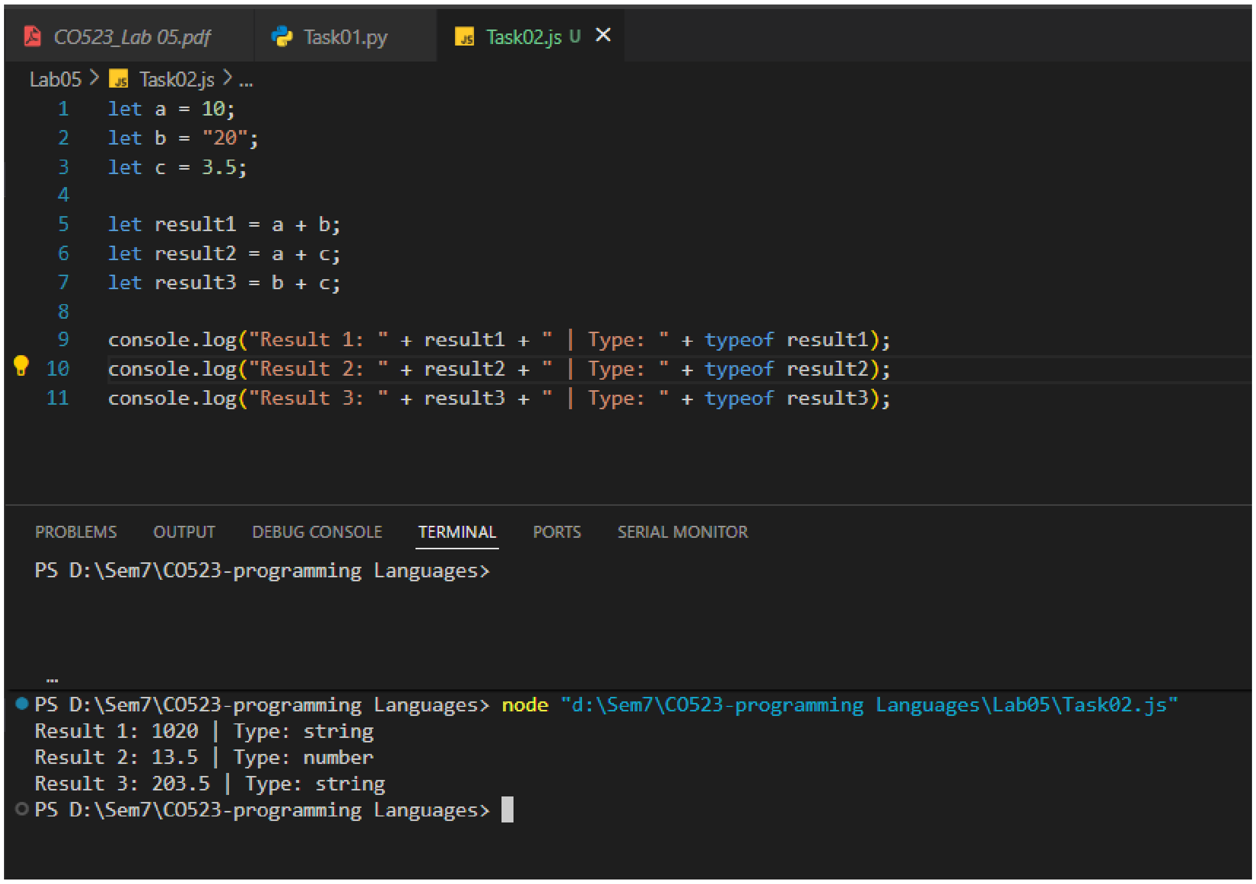
- The code executes the loop and checks the types of data[i] and data[j] only when it reaches the addition line during execution.

- In a statically typed language (like Java), an array must usually contain only one type of data. Python allows a single list (data) to hold an int, str, and float simultaneously, and it only determines if they can be added at the exact moment the program runs.

B. Python is Strongly Typed

- Most mixed-type operations failed with a **TypeError**. For example, when trying to add 1 (int) and "2" (str), Python did not automatically convert the number to a string to produce "12" or the string to a number to produce 3.
- Python strictly enforces that types must be compatible for an operation. It only allowed the addition of int + float because they are both numeric types, but it blocked int + str and float + str because they are fundamentally incompatible without an **explicit conversion**.

Task 02



```javascript
let a = 10;
let b = "20";
let c = 3.5;

let result1 = a + b;
let result2 = a + c;
let result3 = b + c;

console.log("Result 1: " + result1 + " | Type: " + typeof result1);
console.log("Result 2: " + result2 + " | Type: " + typeof result2);
console.log("Result 3: " + result3 + " | Type: " + typeof result3);
```

```
PS D:\Sem7\CO523-programming Languages>

...
PS D:\Sem7\CO523-programming Languages> node "d:\Sem7\CO523-programming Languages\Lab05\Task02.js"
Result 1: 1020 | Type: string
Result 2: 13.5 | Type: number
Result 3: 203.5 | Type: string
PS D:\Sem7\CO523-programming Languages>
```

Observed Output

When you execute this code in a browser console or a Node.js environment, you will see:

1. Result 1: 1020 | Type: string
2. Result 2: 13.5 | Type: number
3. Result 3: 203.5 | Type: string

Explanation and Classification

Based on these results:

A. JavaScript is Dynamically Typed

- **Evidence:** Variables a, b, and c are declared using let without specifying a type (like int or String).
- **Reasoning:** The language determines the type of the variable at **runtime** based on the value assigned to it. Furthermore, variables in such systems can hold values of different types at different times without causing a compilation error.

B. JavaScript is Weakly Typed

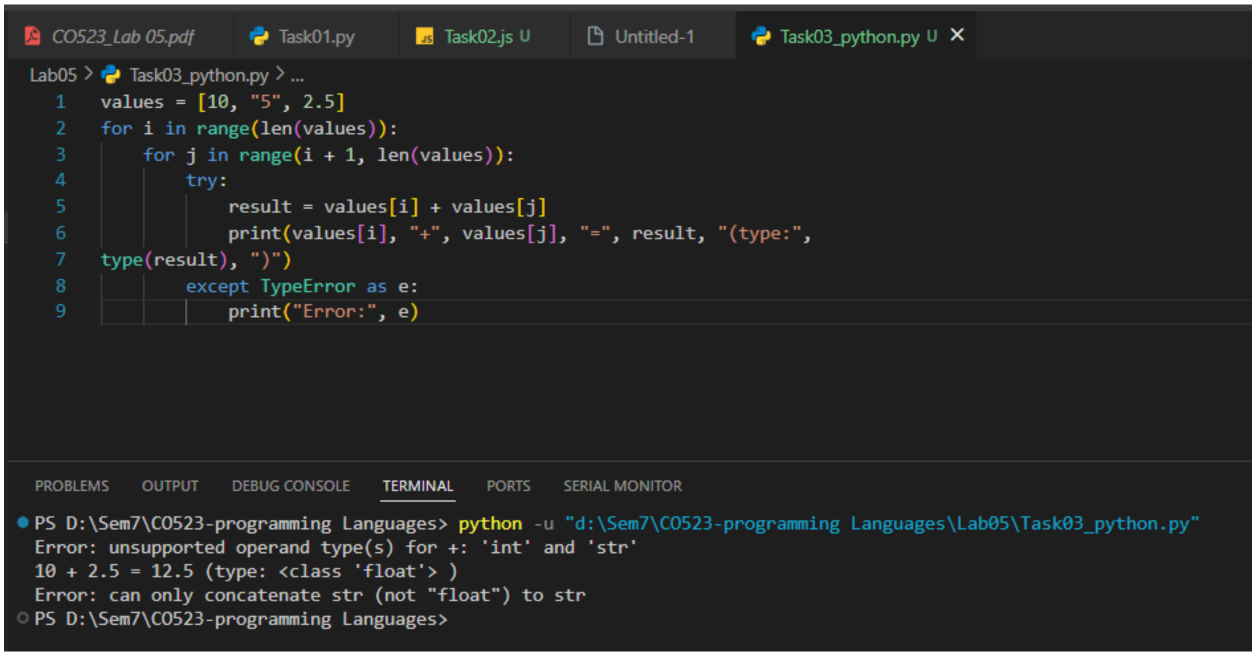
- **Evidence:** * **Result 1 (a + b):** The number 10 was implicitly converted to the string "10" and then concatenated with "20" to produce "1020".
    - **Result 3 (b + c):** The number 3.5 was implicitly converted to the string "3.5" to concatenate with "20", resulting in "203.5".
- **Reasoning:** Weak typing allows operations between incompatible types through **implicit (automatic) type conversion** (also known as coercion). Unlike Python (which would throw a TypeError), JavaScript "coerces" the values to a compatible form to complete the operation.

Analysis of Practical Implications

Feature	**Advantage**	**Disadvantage**
Weak Typing	**Flexibility -** Allows for faster development and less verbose code because you don't always have to manually convert types.	**Unpredictability -** Can lead to "silent" logical errors. For example, a developer might expect 10 + "20" to be 30, but getting "1020" could break later logic without crashing the program.
Dynamic Typing	**Rapid Prototyping -** Easier to write and change code quickly since you don't need to manage strict type declarations.	**Performance & Safety -** Can reduce execution performance and increase the risk of errors that only appear when the program is already running

Task 03

Step 1: Run and Observe Initial Programs
Python Program

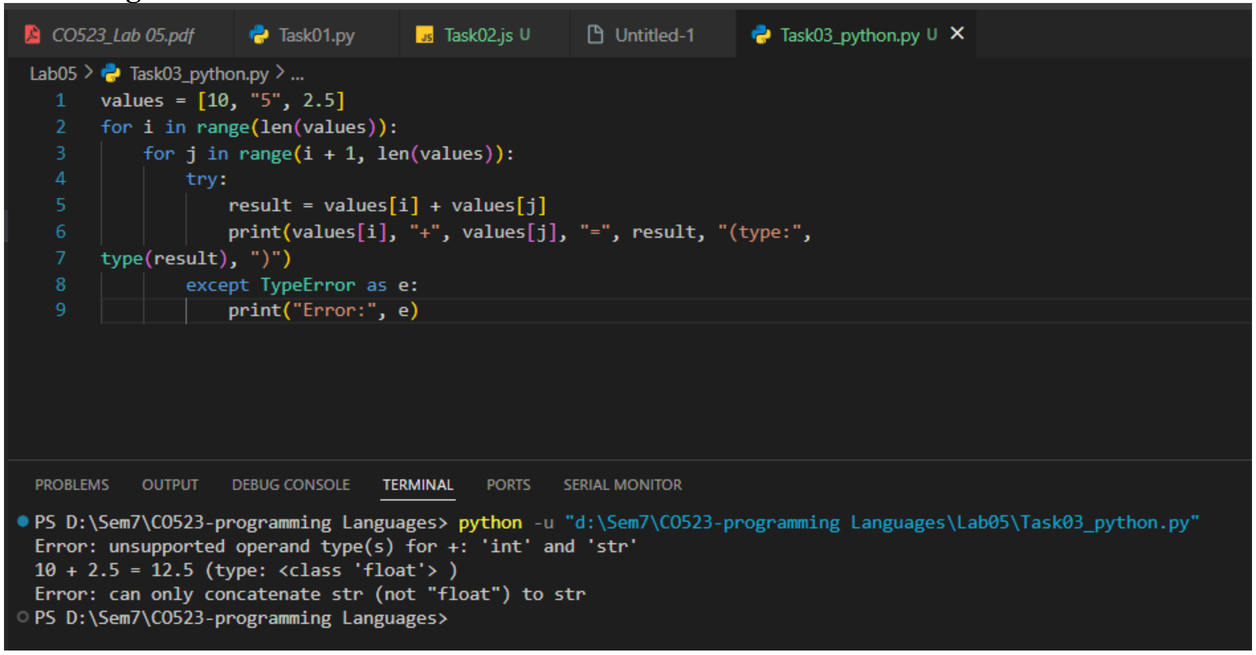


```python
values = [10, "5", 2.5]
for i in range(len(values)):
    for j in range(i + 1, len(values)):
        try:
            result = values[i] + values[j]
            print(values[i], "+", values[j], "=", result, "(type:",
type(result), ")")
        except TypeError as e:
            print("Error:", e)
```

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\Lab05\Task03_python.py"
Error: unsupported operand type(s) for +: 'int' and 'str'
10 + 2.5 = 12.5 (type: <class 'float'> )
Error: can only concatenate str (not "float") to str
PS D:\Sem7\CO523-programming Languages>
```

- **Result 1 (10 + 2.5):** Succeeds. Output: 10 + 2.5 = 12.5 (type: <class 'float'>).
- **Result 2 (10 + "5" or "5" + 2.5):** Fails. Output: Error: unsupported operand type(s) for +: 'int' and 'str'.

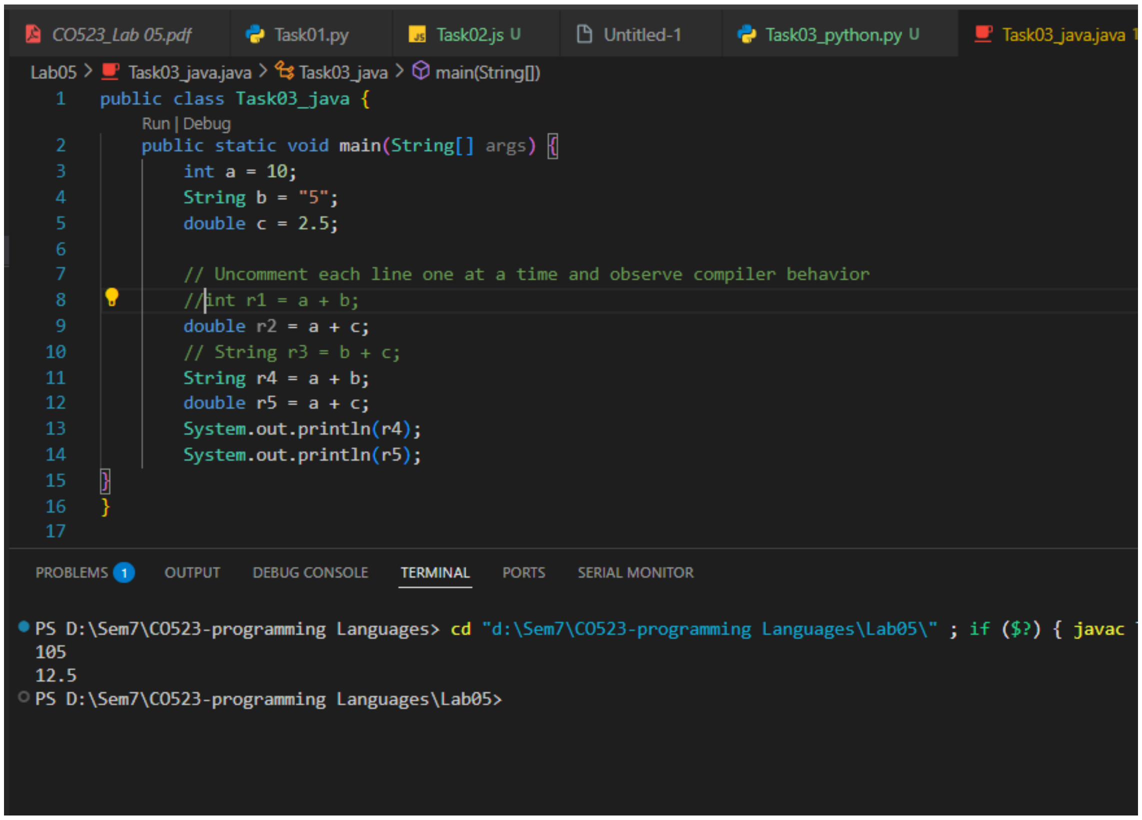
Java Program



```python
values = [10, "5", 2.5]
for i in range(len(values)):
    for j in range(i + 1, len(values)):
        try:
            result = values[i] + values[j]
            print(values[i], "+", values[j], "=", result, "(type:",
type(result), ")")
        except TypeError as e:
            print("Error:", e)
```

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\Lab05\Task03_python.py"
Error: unsupported operand type(s) for +: 'int' and 'str'
10 + 2.5 = 12.5 (type: <class 'float'> )
Error: can only concatenate str (not "float") to str
PS D:\Sem7\CO523-programming Languages>
```

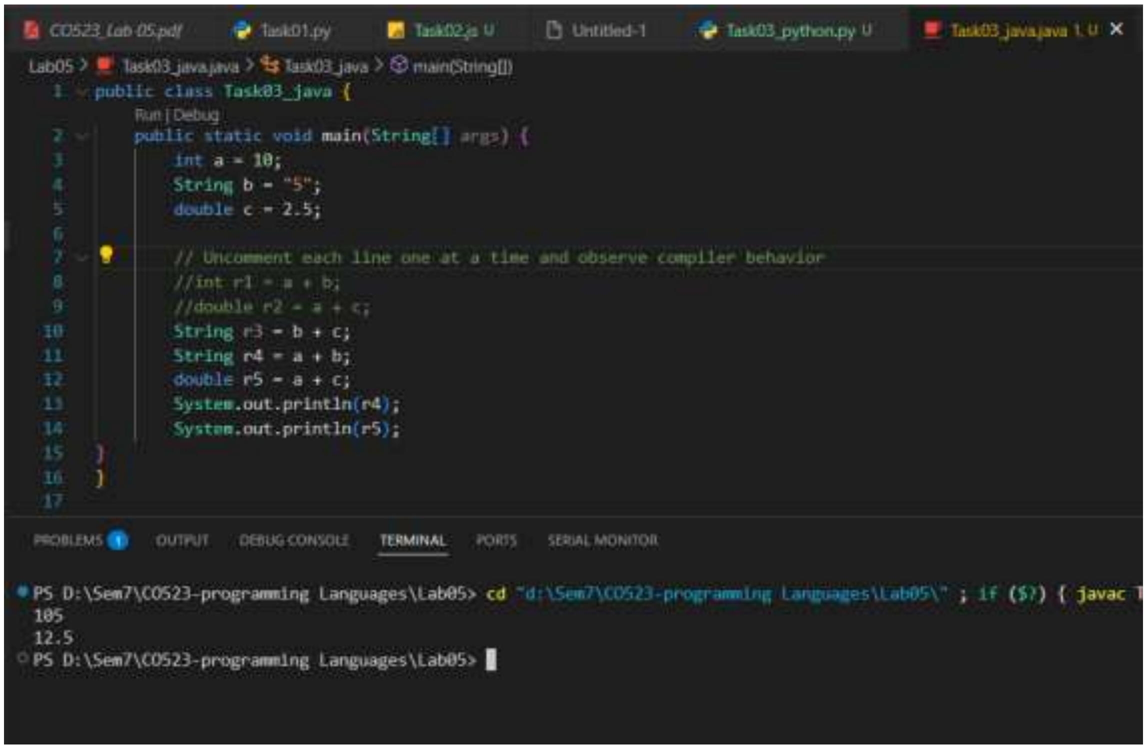
- int r1 = a + b; **Compile Error:** Incompatible types; String cannot be converted to int.



```java
public class Task03_java {
    public static void main(String[] args) {
        int a = 10;
        String b = "5";
        double c = 2.5;

        // Uncomment each line one at a time and observe compiler behavior
        //int r1 = a + b;
        double r2 = a + c;
        // String r3 = b + c;
        String r4 = a + b;
        double r5 = a + c;
        System.out.println(r4);
        System.out.println(r5);
    }
}
```

```
PS D:\Sem7\CO523-programming Languages> cd "d:\Sem7\CO523-programming Languages\Lab05\" ; if ($?) { javac
105
12.5
PS D:\Sem7\CO523-programming Languages\Lab05>
```

- double r2 = a + c; **Success:** Compiles and runs. Java performs **implicit promotion** of int to double.



```java
public class Task03_java {
    public static void main(String[] args) {
        int a = 10;
        String b = "5";
        double c = 2.5;

        // Uncomment each line one at a time and observe compiler behavior
        //int r1 = a + b;
        //double r2 = a + c;
        String r3 = b + c;
        String r4 = a + b;
        double r5 = a + c;
        System.out.println(r4);
        System.out.println(r5);
    }
}
```

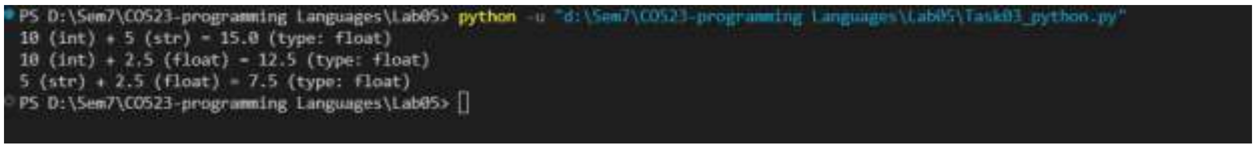
```
PS D:\Sem7\CO523-programming Languages\Lab05> cd "d:\Sem7\CO523-programming Languages\Lab05\" ; if ($?) { javac
105
12.5
PS D:\Sem7\CO523-programming Languages\Lab05>
```

- String r3 = b + c; **Success:** Compiles and runs. Java performs **string concatenation** by converting the number to a string.

Modify for Explicit Conversions
To make the failing operations work, we must apply **explicit type conversion** (casting or parsing).

Modified Python

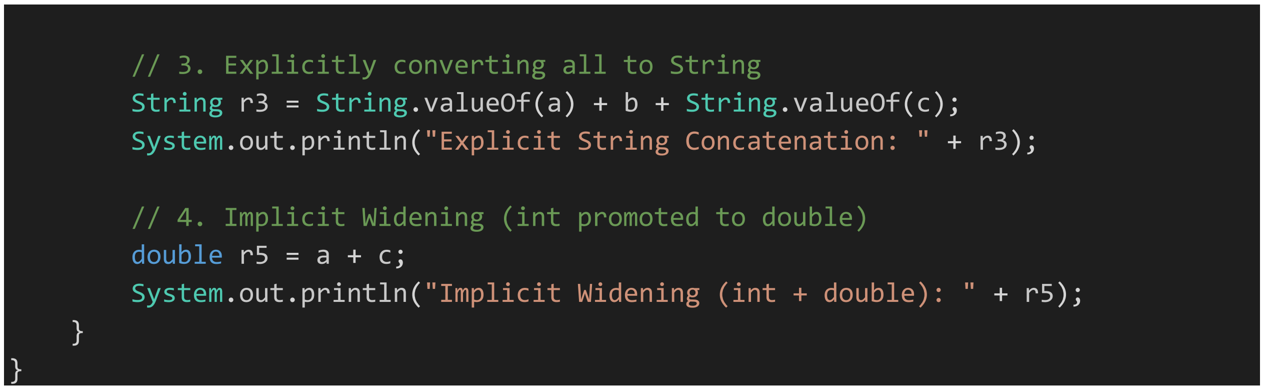
```python
values = [10, "5", 2.5]

for i in range(len(values)):
    for j in range(i + 1, len(values)):
        try:
            val1 = values[i]
            val2 = values[j]

            # Explicitly converting both operands to float to ensure mathematical
addition
            # This handles mixtures of int, string-numbers, and floats safely.
            result = float(val1) + float(val2)

           print(f"{val1} ({type(val1).__name__}) + {val2} ({type(val2).__name__})
= {result} (type: {type(result).__name__})")
        except (TypeError, ValueError) as e:
            print(f"Error adding {values[i]} and {values[j]}: {e}")
```



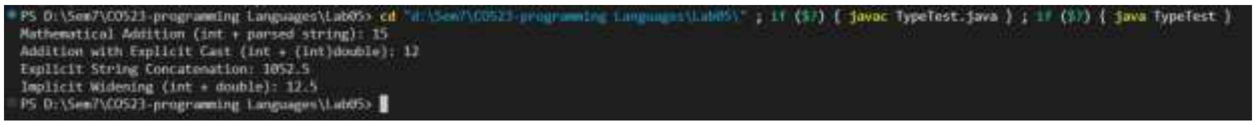
```
PS D:\Sem7\COS23-programming Languages\Lab05> python -u "d:\Sem7\COS23-programming Languages\Lab05\Task03_python.py"
10 (int) + 5 (str) = 15.0 (type: float)
10 (int) + 2.5 (float) = 12.5 (type: float)
5 (str) + 2.5 (float) = 7.5 (type: float)
PS D:\Sem7\COS23-programming Languages\Lab05> []
```

Modified Java

```java
public class TypeTest {
    public static void main(String[] args) {
        int a = 10;
        String b = "5";
        double c = 2.5;

        // 1. Explicitly converting String to Integer for mathematical addition
        int r1 = a + Integer.parseInt(b);
        System.out.println("Mathematical Addition (int + parsed string): " + r1);

         // 2. Explicitly casting double to int (Narrowing Conversion - loses
precision)
        int r2 = a + (int)c;
         System.out.println("Addition with Explicit Cast (int + (int)double): " +
r2);
```



```
        // 3. Explicitly converting all to String
        String r3 = String.valueOf(a) + b + String.valueOf(c);
        System.out.println("Explicit String Concatenation: " + r3);

        // 4. Implicit Widening (int promoted to double)
        double r5 = a + c;
        System.out.println("Implicit Widening (int + double): " + r5);
    }
}
```



```
PS D:\Sem7\CO523-programming Languages\Lab05> cd "d:\Sem7\CO523-programming Languages\Lab05\" ; if ($?) { javac TypeTest.java } ; if ($?) { java TypeTest }
Mathematical Addition (int + parsed string): 15
Addition with Explicit Cast (int + (int)double): 12
Explicit String Concatenation: 1052.5
Implicit Widening (int + double): 12.5
PS D:\Sem7\CO523-programming Languages\Lab05>
```

Comparison and Evidence

Feature	**Python**	**Java**
Typing Category	**Dynamic Typing** - Types are checked at runtime	**Static Typing** - Types are checked at compile time.
Enforcement	**Strong Typing** - Blocks int + str without explicit conversion.	**Strong Typing** - Generally blocks incompatible assignments (e.g., int x = "hello").
Type Checking	Occurs during program execution (Runtime).	Occurs during compilation. Errors prevent the program from running.
Implicit Conversion	Limited (e.g., int to float). Highly restrictive with strings.	Broad for strings (concatenation) and numeric promotion (widening).

Advantages and Disadvantages

- **Static Typing (Java):**
    - **Pros:** Detects errors early before deployment. Better performance due to compiler optimizations.
    - **Cons:** More "boilerplate" code; you must declare every variable's type.
- **Dynamic Typing (Python):**
    - **Pros:** Faster to write and more flexible; code is less verbose.
    - **Cons:** Errors might stay hidden until a specific line of code is executed at runtime.
- **Explicit Conversion Requirement:**
    - **Advantage:** Prevents logical bugs (like accidentally adding a zip code string to a price integer).
    - **Disadvantage:** Requires more effort from the programmer to write conversion logic.

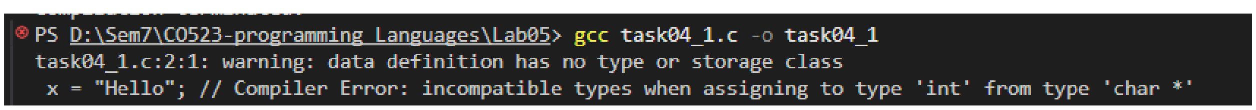
Task 04

1. Is C Statically or Dynamically Typed?

C is a **statically typed** language.

- In C, the type of every variable must be declared at the time of definition (e.g., int x;, float y;). The compiler determines and checks these types during the **compilation phase** before the program ever runs.
- **Example:** If you attempt to assign a string literal directly to an integer variable, the compiler will generate an error or a severe warning during build time, preventing the creation of an executable.

```
int x;
x = "Hello"; // Compiler Error: incompatible types when assigning to type 'int'
from type 'char *'
```



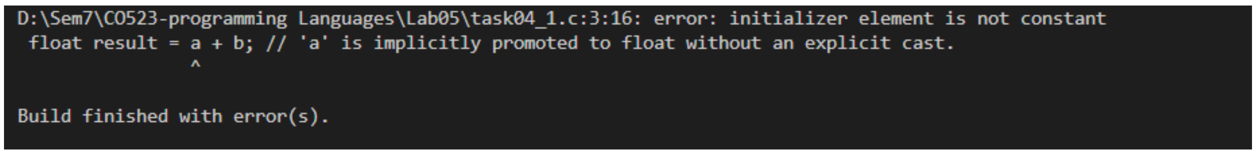
```
PS D:\Sem7\CO523-programming Languages\Lab05> gcc task04_1.c -o task04_1
task04_1.c:2:1: warning: data definition has no type or storage class
 x = "Hello"; // Compiler Error: incompatible types when assigning to type 'int' from type 'char *'
```

2. Is C Strongly or Weakly Typed?

C is generally classified as a **weakly typed** (or loosely typed) language.

- **Explanation:** While C has a type system, it is not strictly enforced compared to languages like Python or Java. C allows for **implicit type conversions** and provides mechanisms like pointers and void* that can be used to bypass type checks.
- **Example (Implicit Coercion):** C will often silently convert types to make an operation work rather than throwing an error.

```
int a = 10;
float b = 5.5;
float result = a + b; // 'a' is implicitly promoted to float without an explicit
cast.
```



```
D:\Sem7\CO523-programming Languages\Lab05\task04_1.c:3:16: error: initializer element is not constant
 float result = a + b; // 'a' is implicitly promoted to float without an explicit cast.
                ^

Build finished with error(s).
```

- **Example (Pointer Type Bypassing):** You can use pointers to treat a block of memory as a different type entirely, which a strongly typed language would normally forbid for safety reasons.

```
float f = 3.14;
int *p = (int *)&f; // Forcing an integer pointer to point to a float memory
address.
```