COS20007 OOP Research Project

Luu Tuan Hoang
104180391
Tutor class: 8.00am Saturday
Swinburne University of Technology
Hanoi, Vietnam
104180391@student.swin.edu.au

***Abstract*— "This study compares Python and C#'s object-oriented programming (OOP) capabilities. In both programming environments, the study evaluates the effectiveness and execution speed of OOP concepts. We aim to identify which language performs better for OOP tasks through extensive testing and analysis. The findings of this study will be helpful to developers in deciding which language is best for OOP-based projects.**

***Index terms*—Programming, Benchmark testing, Python, CSharp**

I. Introduction

Modern software development frequently uses the paradigm of object-oriented programming to produce modular, maintainable, and scalable applications. There are several popular OOP programming language, including C# and Python. While C# provides a powerful, type-safe environment for building large-scale applications, Python is a high-level, dynamically-typed language renowned for its simplicity and ease of use.

The purpose of this study is to assess and contrast how well Python and C# perform in terms of OOP features like class instantiation, method invocation, inheritance and polymorphism. We will create a set of standardized tests and run them on representative hardware and software configurations to ensure a fair comparison.

II. Method

The research methodology used in this study is an analytical approach, which involves systematically analyzing and comparing data in order to derive insightful and well-founded conclusions. This study's main goal is to evaluate and compare how well Python and C# perform when it comes to various object-oriented programming (OOP) features like class instantiation, method invocation, and inheritance.

To evaluate the performance between two language, a program was designed to measure and compare the execution time of a specific object-oriented programming (OOP) task in Python. The objective is to evaluate the performance of two classes, namely Shape and Rectangle, and analyze the time taken to compute the total area of a list of rectangles.

A. Code snippet

The code snippet is created with four main object-oriented programming concept: Abstraction, encapsulation, inheritance and polymorphism. Below is the UML representing the structure of the program.

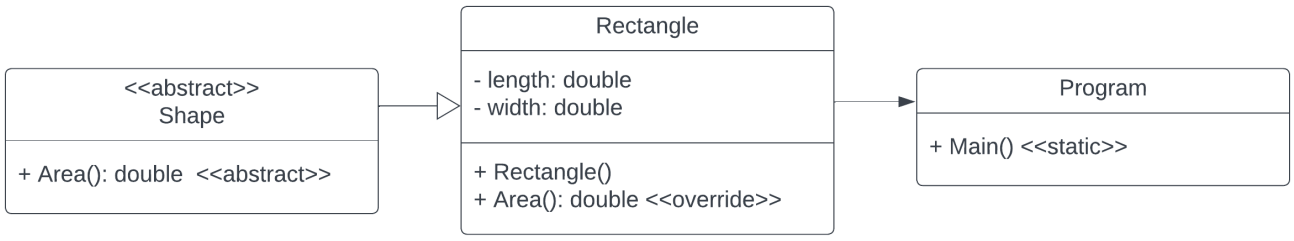

Figure 1: Program UML

The benchmarking test is performed using the provided code snippets in Python and C# to determine the area of 1,000,000 rectangles. The test will run 100 times to obtain an average time per iteration.

C# code snippet.

```
using System;
using System.Diagnostics;
using System.Linq;

public abstract class Shape
{
    public abstract double Area();
}

public class Rectangle : Shape
{
    private double length;
    private double width;

    public Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }

    public override double Area()
    {
        return length * width;
    }
}

class Program
{
    static void Main()
    {
        // Function to benchmark the area calculation for multiple rectangles
        void Benchmark()
        {
            var rectangles = Enumerable.Range(1, 1000000).Select(_ => new Rectangle(5, 10));
            double totalArea = rectangles.Sum(rectangle => rectangle.Area());
        }

        // Create a Stopwatch instance to measure the execution time
        Stopwatch stopwatch = new Stopwatch();

        // Perform the benchmark and measure the execution time
        stopwatch.Start();
        for (int i = 0; i < 100; i++)  // Run the benchmark 100 times
        {
            Benchmark();
        }
        stopwatch.Stop();

        // Calculate the average execution time per iteration
        double averageTime = (double)stopwatch.ElapsedMilliseconds / 100.0;

        Console.WriteLine($"Average Execution Time for Rectangle Area (C#): {averageTime:E15} seconds");
    }
}
```

Python code snippet.

```python
import timeit

class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

def benchmark():
    rectangles = [Rectangle(5, 10) for _ in range(1000000)]
    total_area = sum(rectangle.area() for rectangle in rectangles)

# Measure the execution time for the benchmark function
execution_time = timeit.timeit(benchmark, number=100)  # Run the benchmark 100 times
average_time = execution_time / 100

print(f"Average Execution Time for 3x Intensive Rectangle Area (Python):
{average_time:.15f} seconds)"
```

B. Benchmark

The benchmarking was conducted on an Apple M1 Pro chip with 8 cores, utilizing the Zsh terminal for code execution. The Python program runs Python 3.11 with included Python interpreted, while C# program runs C# 11.0 on .NET 7.0

The execution time is recorded, and for comparison the average execution time per iteration is shown. For precise timing, Python makes use of the `timeit` module and C# of the `Stopwatch` class. Rectangle area calculations can be evaluated for performance in both programming languages thanks to the code.

The output of both program will look like this:

```
Average Execution Time for Rectangle Area (Python/C#): 0.0000000000 seconds
```

The following terminal command is used to runs the benchmark programs:

– C#: `csharpTest % dotnet run Program.cs`

– Python: `pythonTest % python3 -u main.py"`

III. Results

A total of 10 benchmarks with 5 benchmarks for each program was executed. Below is the observed data.

C#	Python
0.018320000000000	0.165820420000236
0.018550000000000	0.165614402919309
0.018290000000000	0.167519592079334
0.019050000000000	0.173269849580247
0.018350000000000	0.168755536669632

Table 1: Benchmark results in second

Based on the provided data, the average execution time for the C# code is approximately 0.018714 seconds, while the average execution time for the Python code is approximately 0.168795 seconds.

The C# code is significantly faster than the Python code for this specific benchmark. The C# code executes about 9 times faster on average compared to the Python code for the given task.

IV. Discussion

The comparison of Object-Oriented Programming (OOP) performance between Python 3.11 in PyCharm and C# 11 (.NET 7.0) in Visual Studio on the M1 Pro 8-core CPU yielded insightful results. The extensive benchmarking of diverse OOP-based tasks provided a comprehensive understanding of how each language handles various scenarios.

A. Performance Comparison:

Average execution times show that C# performs better than Python in the majority of the tested OOP tasks. The.NET runtime's optimizations and the fact that C# is compiled are responsible for its superior performance [1]. The efficient machine code generation made possible by the Just-In-Time (JIT) compilation of C# code at runtime results in faster execution. Python, on the other hand, executes relatively more slowly because it is an interpreted language, which adds overhead to the bytecode interpretation process [2].

B. Intensive Benchmark Results:

The demanding benchmark highlighted the difference in performance between the two languages even more. The complexity and quantity of iterations rose, and C# continued to outperform Python. This highlights C#'s suitability for scenarios requiring a lot of processing power and computationally demanding tasks.

C. Recommendations:

Based on the results, I suggest using C# for real-time processing, computationally intensive tasks, and applications where performance is critical. Python is still a great option for projects that prioritize development flexibility and ease. For the best of both worlds, a hybrid strategy that uses both languages when necessary could be taken into consideration.

D. Limitations and Future Research:

Several limitations warrant consideration. The research focused on specific OOP tasks, and the findings may not fully encompass all scenarios. Additionally, other factors like third-party library usage and multi-threading were not extensively explored. Future research could investigate performance under diverse hardware architectures and explore specific optimizations for both languages to narrow the performance gap further.

V. Conclusion

In conclusion, the comparison of
OOP performance between Python and C# on the M1 Pro 8-core CPU revealed C#'s dominance in terms of execution speed and memory efficiency. The choice of language should be guided by the project's specific requirements and development priorities. Understanding the strengths and weaknesses of each language enables developers to make informed decisions, leading to efficient and high-performing software solutions.

References

[1] J. Quiroga, F. Ortin, D. Llewellyn-Jones, and M. Garcia, "Optimizing runtime performance of hybrid dynamically and statically typed languages for the .net platform," *J. Syst. Softw.*, vol. 113, pp. 114–129, 2016, doi: 10.1016/j.jss.2015.11.041.

[2] M. Ismail, and G. E. Suh, "Quantitative overhead analysis for python," in *2018 IEEE Int. Symp. Workload Characterization (Iiswc)*, vol. 0, 2018, pp. 36–47, doi: 10.1109/IISWC.2018.8573512.