An empirical performence evaluation between Python and Scratch

Morgan Norderg, Linköping University

1 Abstract

In this empirical study the authors examine and evaluate the performence of Scratch and Python for the sum-for function, and find that there is a good indication that Python has better performence characteristics compared to Scratch regarding iterations. The authors conclude that Scratch appears to be an unattractive choice for high performence programming, however they state that further research is needed.

2 Introduction

Python is a widely adopted high level multiparadigm programming language in todays tech industry. It is however well known to be lacking in performence, and thus there is a natural interest in potentially high performing languages that could serve as fitting replacements. Scratch offers many of the same benefits as Python, such as a easy learning curve, interpreted nature allowing for iterative development, and wide adoption in programming education. So investigating it's performence is imperative for the advancement of the field of computer science and the tech industry as a whole.

3 Background

Scratch was initially developed at Massacheussets institute of technology(MIT) to entertain children. It offers a wide range of features such as an interpreted development enviorment with colorful blocks for visual programming. It's performence characteristics have been unknown and uncharted territory, until today!

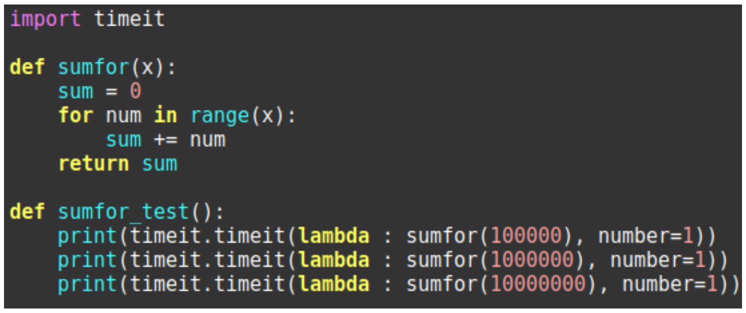
Python is a interpreted, dynamically typed, programming language introduced in 1994. It is today deployed in all areas of the tech industry, from data analysis and AI, to crypto mining and OS development(probably).

Iteration with for-loops in Python is notoriously slow, so a major point of this paper is to investigate if Scratch offers a compelling alternative for these use-cases. Since Scratch is interpreted in Javascript(Random friend of the author, 2024), which is generally faster than Python, it is hypothesized by the author that Scratch might outperform Python in this area.

The sum-for function is a function that sums the integer values of 0 to n.

4 Related work

There appears to be no related work on the subject. The authors didn't look for any, and found nothing.

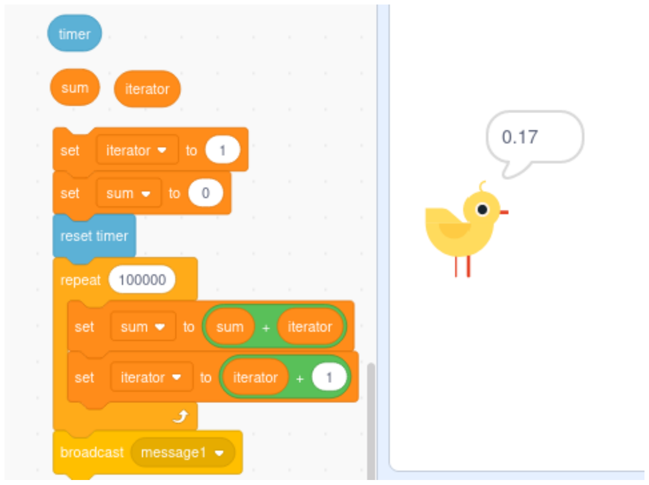


```python
import timeit

def sumfor(x):
    sum = 0
    for num in range(x):
        sum += num
    return sum

def sumfor_test():
    print(timeit.timeit(lambda : sumfor(100000), number=1))
    print(timeit.timeit(lambda : sumfor(1000000), number=1))
    print(timeit.timeit(lambda : sumfor(10000000), number=1))
```

Figure 1: Python source code used for collectiong the measurements



```
timer
sum  iterator
set iterator to 1
set sum to 0
reset timer
repeat 100000
    set sum to sum + iterator
    set iterator to iterator + 1
broadcast message1
```

Figure 2: Scratch source code used for collecting the measurements

5 Methodology

For comparing the iteration performence between Scratch and Python, we compared the time in seconds for running a simple sum-for function, for the input values of 100-000, 1-000-000 and 10-000-000. To collect the measurements for the Python code the authors ran the Python script n=1 times and reported the average runtime in seconds. n=1 was choosen because that's how many times the author could be bothered to run the Scratch code. For outputting the results the Python script used the Print function of the corresponding standard library, meanwhile the Scratch code used a cartoon chicken. The cartoon chicken is nessecary since the say-block in Scratch is only available for sprites in Scratch.

The broadcast block shown in figure 2 sends a message to a recieve broadcast block in the chicken sprite logic, not shown in this study.

6 Results

The results showcase an massive performence decrease with Scratch compared to Python. In table below we can see the time in seconds for performing n iterations of the sum-for function in Python and Scratch respectivley.

Python	Scracth	n
0.003	0.200	100-000
0.032	1.96	1-000-000
0.318	15.91	10-000-000

7 Discussion

It's rather dissapointing to the author that Python outperformed Scratch to such a significant extent. This runs contrary to the authors previously stated hypothesis. There are however some potential factors that could have influenced the results.

7.1 Scratch optimizations

It is possible that the overhead in the Scratch code could be reduced by moving the sum-for function code block directly into the logic of the cartoon chicken sprite, removing the need to broadcast a message to the chicken sprite to output the result. Another possible way to reduce this overhead the authors hyopthesize, is to store the value of the timer variable in a separate immutable value, and say that variable value instead in the chicken sprite. The overhead then would simply be the cost of an assigment operation which should be insignificant.

7.2 Unknown factors

It is not exactly known to the authors how the timeit function in Python works under the hood, and there might be some implementation details that create less overhead compared to the timer in Scratch. The authors did not investigate this due to a severe lack of f***s to give.

8 Further work

Due to the stated importance of the subject, the lack of pre-existing sientific litterature regarding the subject, and the preiously stated potential for Scratch optimizations there is a clear need for further research.

8.1 Scratch optimizations

Besides replicating this research, alongside the previously stated optimization gains to further solidify our understanding of Scratch performence characteristics; there is a potential area of research to explore regarding Scratch to Scratch compiler technology, or perhaps transpiling Scratch code to highly performant C code. Or highly performant Rust code, since the Biden administration outlawed the C programming language in 2024 *("Press Release: Future Software Should Be Memory Safe", whitehouse.gov, 2024-02-26, https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/).*
There is a notable lack in research regarding the development of compilitation, transpilation and genetic mutation of Scratch code, and other forms of build tools. This could potentially revolutionize the world of high performenece Scratch programming. There is also a possibility of looking into hardware accelerated Scratch programming using hardware description languages and FPGA's or ASICS.

8.2 Use cases

Since this study only considered the case of the sum-for function, there could be interest in looking at other potential use cases of the Python programming language where Scratch might possibly outperform it. We are deperatley in need of further research in the area of Scratch driven OS-kernal development.

9 Conclusions

You probably don't wanna use Scratch for performence senitive programming tasks.

References

https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/