Performance comparison of Julia and Python programming languages based on available literature

M. MAJ
mateuszmaj2@mon.gov.pl

Dowództwo Komponentu Wojsk Obrony Cyberprzestrzeni
Gen. Broni T. Buka 1, 05-119 Legionowo, Poland

The Julia language is a fast-growing high-level programming language. Its developers suggest that the quality and implementation of innovative compilation technology (Just-In-Time Compilation) beats Python in terms of algorithm execution times. The selection of the right programming language to implement an algorithm is of paramount importance. In today's world every millisecond saved can determine the success of a product. The choice has a direct impact on the performance of the application or the execution of the algorithm. Consequently, a comparison analysis of modern or emerging technologies becomes particularly important. This article is the first part of a comparative analysis, the continuation of which [21] expands the research on the subject of this paper. In this context, the article presents summary of the existing literature and discusses the time performance of Julia and Python programming languages. In addition, test results and published source codes were reproduced and verified in a research environment. In the second part of the article [21], the research was extended to include more algorithms and to check the performance of each language on graphics cards.

Keywords: Python, Julia, comparison, performance.

DOI: 10.5604/01.3001.0055.2084

1. Introduction

1.1. Background of the research

According to Guido van Rossum, the Python language was derived from popular and widely used programming languages such as ABC. Python was designed with the objective of enhancing programming standards, thereby paving the way for new ones [1].

Julia language also derives from well--established and widely recognized programming languages such as C, R and the aforementioned Python. It builds its strength on their eliminating the trade-off problem, such as the simplicity of Python, which comes at the cost of performance issues. Another example is the C language, which excels in speed but has a complex syntax that makes it significantly more difficult to use.

The implementation of the Julia language dates back to 2009 when Jeff Bezanson, Alan Edelman, Stefan Karpiński and Viral B. Shah began working on a new programming language. Their goal was to develop a tool that would support researchers and scientists in the field of technical computing. Julia was created as a compromise between the performance of compiled languages and the simplicity and flexibility of scripting languages. The name of the language was selected by chance. The Authors found the name "Julia" simply pretty and, as they declare this was the only reason for choosing this as their programming language name [2], [3].

The semantics of Julia are based on the latest Just-In-Time (JIT) compilation technology, compiling code just before execution. JIT compilation is a method of compiling code at runtime rather than using standard compilation before running the program. The JIT system perpetually analyses code during execution by identifying fragments. This specific solution combines the advantages of traditional pre-execution compilation and interpretation to achieve the speed of compiled code (while retaining the flexibility offered by interpreted languages) [4], [15].

All abbreviations and simplified terms employed in this manuscript are fully expanded and defined in the "List of Commonly Used Abbreviations and Terms" provided at the end of the document. The core of this article is a comparative analysis of contemporary literature on both languages, with the aim of identifying the more efficient tool. Authors' reported results were verified were it was possible. In addition, studies that provide source code and input data were repeated in computer environment.

This article has follow-up [21] in the form of extending the research to check the temporal performance of languages using different algorithms on the CPU and GPU. The work has been divided in order to meet publishing criteria.

1.2. Specification of the computing environment

All of the aforementioned below repeated test were conducted using the computer environment described in Table 1. Should alternative versions of the mentioned tools be used, this will be noted in the text.

Tab. 1. Description and specification of the computing environment

Operating system name	Microsoft Windows 10 Home 22H2
System type	x64-based PC
Device model	HP Pavilion Notebook 15-bc5xxx
Processor	Intel(R) Core (TM) i5-9300H CPU @ 2.40GHz, 2400 MHz, Cores: 4, Logic processors: 8
Installed physical memory (RAM)	16,0 GB
Motherboard manufacturer and product	HP 8640
Python environment	Python 3.11.2
Julia environment	Julia 1.8.5
Graphics card	NVIDIA GeForce GTX 1050
Version of the execution environment	Visual Studio Code 1.79.0
Internet connection	NO ACCESS

The development environment used for taking measurements and calling programs is Visual Studio Code. It is a free program that is a customizable editor. The Julia and Python languages are officially supported in VS Code (Visual Studio Code). The tool enables compiling, executing, and debugging source code, including files with the standard extensions for each language i.e. .jl for Julia and .py for Python. The IDE has a wide range of capabilities for reading files, editing code, and creating documentation. In addition, it is a much more complex tool beyond the scope of use to Python and Julia languages alone. VS Code can be linked to a remote repository such as GitHub or the Git version control system [6], [7], [8], [9].

1.3. Research justification

After a dozen or so years since its publication, the Julia language has become recognized almost all over the world. It is supported by a constantly growing and extremely helpful community sharing huge amounts of knowledge [5]. The success of Julia has a major impact on its dissemination thus it is necessary to verify whether this language properly meets the declarations of its creators in terms of code executing speed checking by time efficiency. Is Julia significantly faster than an interpreted language such as Python? Is it as fast as is language's creators declare? A comparison based on the available literature will track the performance of both languages when performing basic operations (White [5]). When working on complex algorithms using high-level libraries (Lin [12], [13]) and when working on very extensive projects (Sells [11]). The literature also provides information on well-known and widely used algorithms (Julia language homepage [14] and Kouatchou's [16], [20]). It is worth taking a closer look at the existing literature in this area of science.

2. Literature review and verification

In order to make a reliable analysis and comparison, it is necessary to look at the existing literature in the field of comparative analysis of the results of the execution times of algorithms in the Julia language relative to contemporary programming languages used on a wide scale such as Python.

It is important, first and foremost, to analyze tests conducted by entities such as the authors of the Julia language. The official tests located on the Julia home page (Benchmarks [14]) were conducted on common algorithms such as 6 random number generators, matrix multiplication or Fibonacci recursions. Simple operations such as writing to a file are also evaluated. The results do not include the execution time of compiling the code for the Julia language. They were conducted for files with native extensions. The operating system used during the tests is openSUSE LEAP 15.0 Linux using a single Intel® Core™ processor core i7-3960X 3.30 GHz with 64 GB of RAM DDR3 1600 MHz [14].

Tab. 2. Summary of results presented on the Julia language home page [14]

Algorithm	Julia time [s]	Python time [s]	Julia / Python [%]
iteration_pi_sum	27,6708	404,3946	6,8
matrix_multiply	70,2494	84,9965	82,6
matrix_statistics	7,3967	80,3211	9,2
parse_integers	0,2177	1,9617	11,1
print_to_file	10,8684	47,0457	23,1
recursion_fibonacci	0,0302	2,1429	1,4
recursion_quicksort	0,2580	9,7296	2,7
userfunc_mandelbrot	0,0527	5,0366	1,0

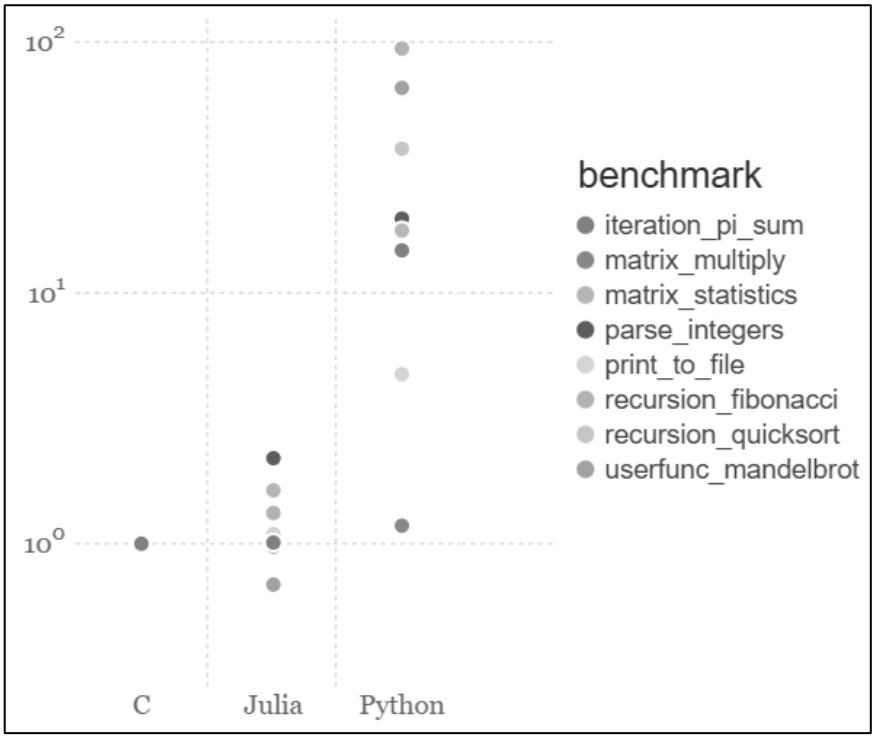

Fig. 1. Graphical representation of results presented on the Julia language home page [14]

The versions of the programming languages used are Julia v1.0.0 and Anaconda Python 3.6.3, respectively. The time results can be observed in the tabular form (Table 2) and on the graph Figure 1 [14].

The analysis presented on the Julia language home page [14] shows that for most algorithms Julia executes on average an order of magnitude faster than Python (Table 2). The exceptions are matrix multiplication and writing to a file. These algorithms were executed in comparable times for both languages. It is also worth noting that the execution time of programs running in Julia is similar to that of programming language C.

The tests described on the official Julia language page [14] were repeated by the author of the paper based on algorithms made available on the GitHub platform by the project owners [10]. To repeat the research, the author of the work used the computer environment described in Table 1. The results of the research conducted in this study are presented in Table 3 and at Figure 2. The table has been enriched with a column containing percentage results (Table 3 with the note – original).

Tab. 3. Summary of the results of repeated tests and presented on the official Julia language website

Algorithm	Julia (Repeated) [s]	Python (Repeated) [s]	Julia / Python (Repeated) [%]	Julia / Python (Original) [%]
iteration_pi_sum	5,1456	356,4379	1,4	6,8
matrix_multiply	19,6119	35,0575	55,9	82,6
matrix_statistics	11,0129	35,8875	30,7	9,2
parse_integers	0,0971	0,9675	10,0	11,1
print_to_file	11,3028	210,0813	5,4	23,1
recursion_fibonacci	0,0325	0,9651	3,4	1,4
recursion_quicksort	0,2757	5,5490	5,0	2,7
userfunc_mandelbrot	0,0507	5,0013	1,0	1,0

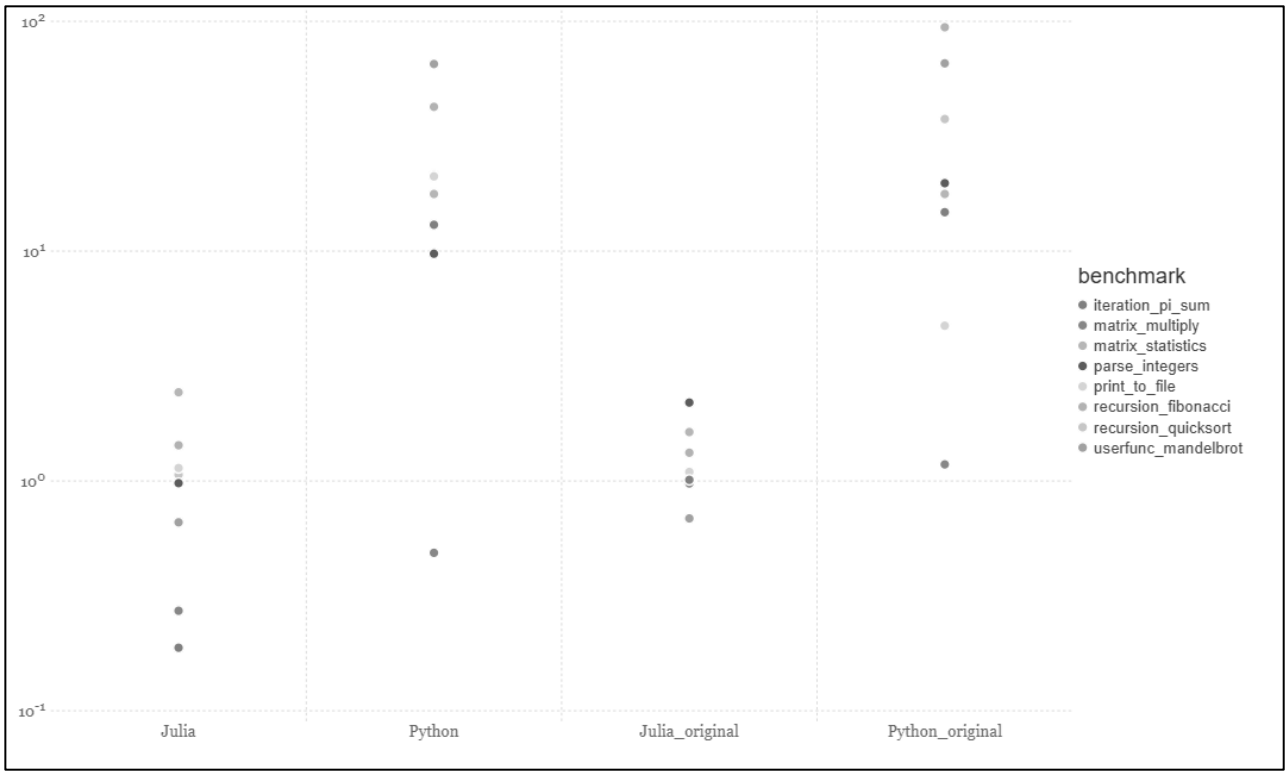

Fig. 2. Image of the results of repeated tests and presented on the official Julia language website

The results of both comparisons for the programs presented in Table 3 are similar, which makes them credible. Figure 2 presents a comparison of the results from Table 2 and Table 3 with the times reported in study [10], normalized to the C language.

The study by the author of the work based on the provided algorithms are similar to those presented on the Julia language homepage. The results shown in Figure 2 were normalized based on the released research on the C language due to the lack of repeated research on this programming language in the research environment described in Table 1. However, it can be inferred that during the repetition of the research as well as during the original tests, Julia outperformed Python.

As Sells [11] reports in his study on missile and rocket simulation, all the language versions are based on a highly developed object oriented simulation architecture adapted to simulate the flight of a missile and a rocket in the time domain. These tests are intended to check the time performance of the Julia language in relation to other advanced or intermediate programming languages (including Python) for a complex type of code.

In order to maintain the efficiency of the programs, Sells delved into the structure of the Julia language by working on several items of literature and reading the official documentation. The results presented in Table 4 were performed on the Windows 10 operating system.

Tab. 4. Summary of results presented by Sells [11]

	Iteration number		
Programming language	1	10	100
C++ time [s]	0,125 (1,00)	0,126 (1,01)	0,127 (1,02)
Python time [s]	3,765 (30,12)	3,752 (30,02)	3,762 (30,10)
Julia time [s]	1,470 (11,76)	0,378 (3,020)	0,269 (2,15)

The used device model is Dell Precision 7820, Intel Xeon Gold 6130 2.10 GHz processor (2 processors with 32 cores and 64 threads). The compilers and language interpreters were used in the basic version without extensions. The language versions used during the study were Julia 1.1.1 and Python 2.7.17, respectively. Two other languages were included in the study, such as Java version 12.0.1 and C++. The tabular results for the Java language will not be provided, except for C++ (Borland 5.5 version) against which the test results were normalized and presented in parentheses [11].

The algorithms were performed in a different number of iterations so that the Julia compiler, which compiles the code during its execution (Just-In-Time), could successively reduce the running time of a single execution and thus provide reliable results. As it can be seen in Table 4, this brought the expected results the time result decreased more than five times for a hundred iterations compared to a single invocation.

The results obtained by the Python language did not change significantly for a larger number of iterations. Figure 3 presents the results from Table 4 in graphical form. Sells also subjected the languages to a his own evaluation based on their coding difficulty. This is illustrated in Figure 3 and is visible on the x-axis[1]. A natural number represents the difficulty level of a language in Sells' subjective assessment – the higher the number, the easier the language it is assigned to. The ordinate presents the time results normalized to the C++ language.

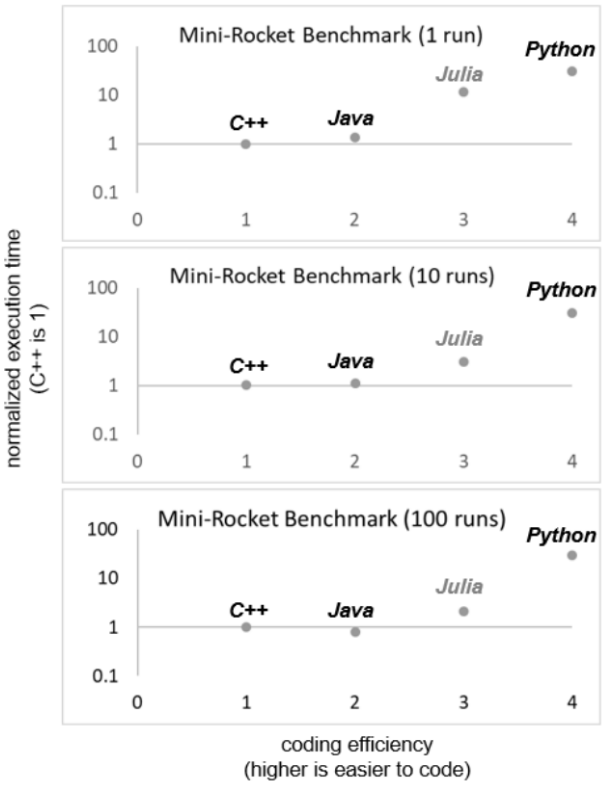

Fig. 3. Results of Sells' research [11]

From Lin's analysis [12], [13] (direct reference to it can be found on the Julia language "Benchmarks" subpage [14] one can deduce the behavior and capabilities of the languages in terms of creating graphs and networks. The tests concern the use of packages implemented in different languages.

Results of Lin's research are presented (Table 5 and Figure 4, The bars correspond to the order listed in the legend) as the median of 100 different single iterations of each command. All packages were tested for three independent datasets (varying in content and number of records) and tested on five different problems: data loading, finding the shortest path from a single source, page ranking, k-core decomposition, and component connection strength [12], [13]. Lightgraphs is the package written in Julia did not perform worse than the tested packages written in Python or its interfaces for almost all tasks (except data loading).

Loading data turned out to be a noticeably big challenge for the Julia language package. The results obtained by this Julia library (Lightgraphs) can be compared to the NetworkX package rejected in further analysis for Amazon and Google data, while for Pokec data it did worse than NetworkX.

[1] The higher the number, the lower the difficulty.

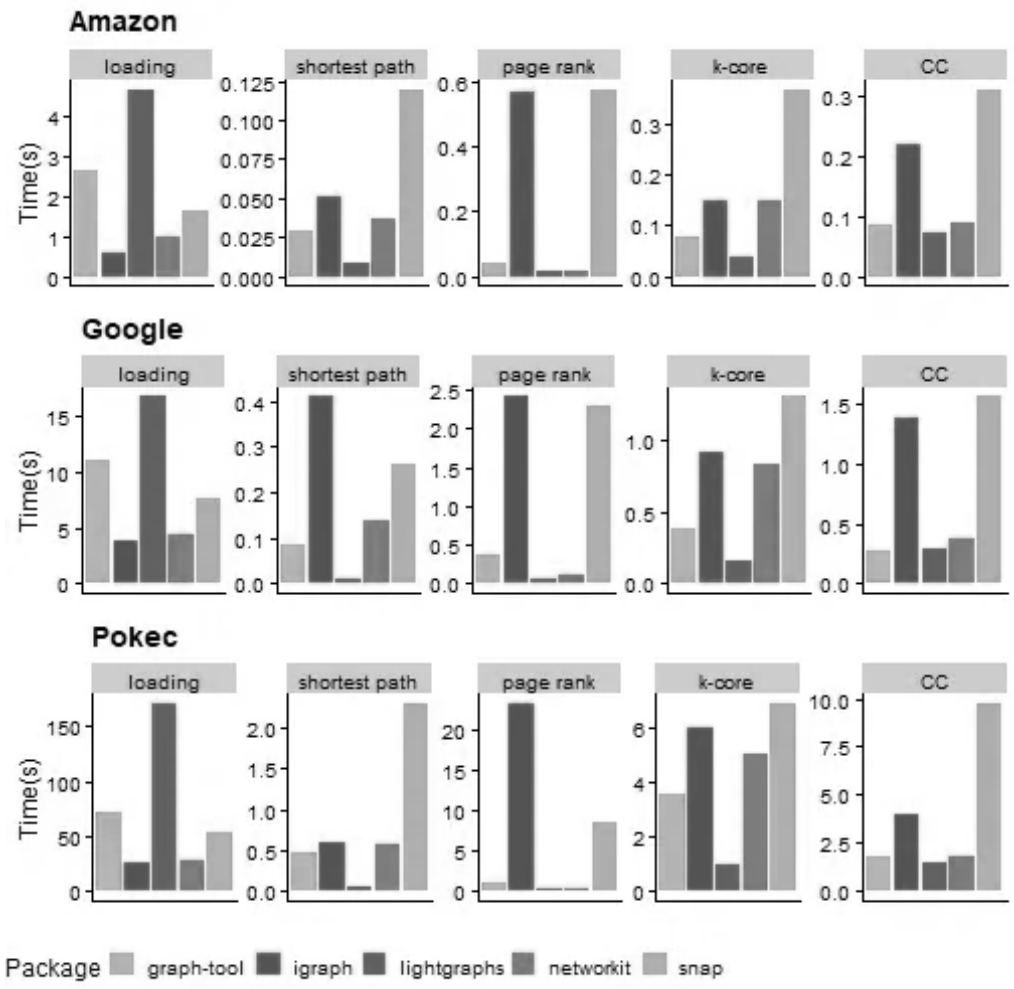

Fig. 4. Results of Lins' research [12]

Tab. 5. Summary of test execution times presented by Lin [12]

		Package name [s]					
Source	**Algorithm**	**graph-tool**	**igraph**	**lightgraphs**	**networkit**	**networkx**	**snap**
Amazon	**CC**	0,09	0,22	0,07	0,09	2,22	0,31
	k-core	0,08	0,15	0,04	0,15	3,63	0,37
	loading	2,61	0,57	4,66	0,98	4,72	1,61
	page rank	0,04	0,57	0,02	0,01	8,59	0,58
	shortest path	0,03	0,05	0,01	0,04	1,37	0,12
Google	**CC**	0,28	1,38	0,29	0,37	7,77	1,56
	k-core	0,39	0,92	0,16	0,83	42,60	1,31
	loading	11,02	3,87	16,75	4,38	19,24	7,56
	page rank	0,36	2,42	0,06	0,10	33,50	9,75
	shortest path	0,08	0,41	0,01	0,14	3,41	0,26
Pokec	**CC**	1,83	3,96	1,50	1,50	61,74	9,75
	k-core	3,60	5,99	0,95	5,02	296,26	6,91
	loading	71,46	25,75	170,63	26,77	140,19	52,73
	page rank	1,10	23,39	0,21	0,24	239,75	8,62
	shortest path	0,48	0,60	0,05	0,56	5,65	2,30

Tab. 6. Summary of the execution times of the tests conducted by White [5]

Result	Julia [s]	Python [s]	Julia / Python [%]
For loop			
Fastest execution	0,00050	0,01098	4,6
Slowest performance	0,00100	0,03098	3,2
Average execution time	0,00059	0,02178	2,7
Conditional operation “if” including text output			
Fastest execution	0,00014	0,01100	1,3
Slowest performance	0,00349	0,02698	12,9
Average execution time	0,00070	0,01479	4,7
Writing out a mathematical equation			
Fastest execution	0,00015	0,00804	1,9
Slowest performance	0,00148	0,02700	5,5
Average execution time	0,00045	0,01529	3,0

Tab. 7. Summary of results of repeated tests presented by White

Result	Julia [s]	Python [s]	Julia / Python [%]
For loop (Repeated)			
Fastest execution	0,00063	0,00100	62,5
Slowest performance	0,00041	0,00097	42,3
Average execution time	0,00055	0,00099	55,2
Conditional operation “if” including text output (Repeated)			
Fastest execution	0,00019	0,00101	18,6
Slowest performance	0,00013	0,00100	12,6
Average execution time	0,00016	0,00100	16,0
Writing out a mathematical equation (Repeated)			
Fastest execution	0,00022	0,00103	21,3
Slowest performance	0,00010	0,00088	11,8
Average execution time	0,00017	0,00099	17,6

According to White [5], testing complex algorithms or even entire applications for individual languages is unreliable. Large programs can be influenced by such programming aspects as coding style and program architecture. White compared and analyzed the fundamental and prevalent operations among contemporary programming languages. These are operations for loop, printing the result of a mathematical operation and the conditional operation “if” containing printing text. The tests were conducted in a computer environment whose specifications correspond to the computer of an average programmer at the time of writing the study. The more important environmental components are an i5 processor and 8 GB of RAM (the operating system is not specified). The results for each of the three programs tested by White are presented in three ways. The results are presented as the most and least favorable, using the average value. The results come from executing the code from ten different iterations on the programs shown as and are presented in Table 6 [5]. In order to verify the results published by White, the author of the work repeated the research carried out by White in the environment described in Table 1.

The times obtained during the aforementioned research were included in Table 6. The results show that the Julia language performs the assigned task unquestionably faster than the Python language, similarly to White’s research [5]. However, the percentage results presented in Table 7 (author’s own work) are less favorable than those presented in 0 (White’s work [5]). The mean results vary by as much as several dozen percentage points. Repeated research shows that Julia performs tasks two to ten times faster, depending on the type of task. The tests conducted by White [5] show that the Julia language performs tasks up to fifty times faster than the Python language.

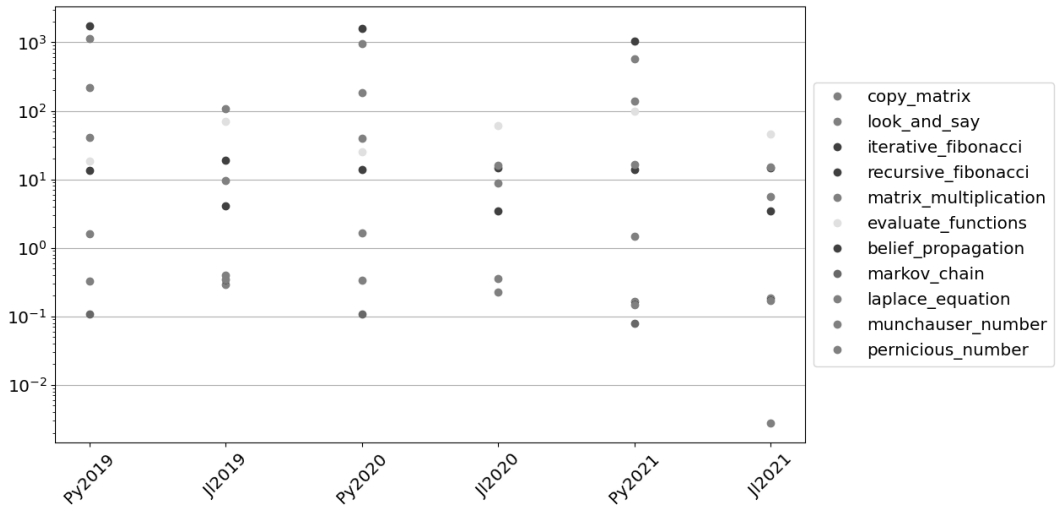

Fig. 5. Results of Kouatchou research [16]

Tab. 8. Summary of the execution times of the tests conducted by Kouatchou [16]

	Python2019 [s]	**Julia2019 [s]**	**Python2020 [s]**	**Julia2020 [s]**	**Python2021 [s]**	**Julia2021 [s]**
copy_matrix	1,6055	0,292153	1,6644	0,2281	1,4590	0,1844
recursive_fibonacci	1735,258	4,126	1573,6125	3,5	1028,6384	3,413
laplace_equation	40,5679	9,5005	40,2565	16,1635	16,6799	14,9538
munchauser	1121,7358	106,2690	968,7658	8,8490	578,3657	5,5290
matrix_multiplication	0,3308	0,350375	0,3344	0,3568	0,1678	0,1693
evaluate_functions	18,2397	69,587	25,3772	60,513	99,7389	45,675
belief_propagition	13,6831	19,2070	14,0236	14,8040	13,8173	14,5380
look_and_say	220,9076	0,4	184,907	0	139,9594	0
iterative_fibonacci	0	0	0	0	0	0
markov_chain	0,1089	0,0	0,1086	0	0,0782	0

Table 8 presents the results published by Kouatchou [16], [20] which are presented in tabular form based on the values provided by the author.

Figure 5 was prepared based on a modified algorithm also provided by the author of the original tests [16]. The abbreviations appearing in the first column of Table 8 and in the legend of Figure 5 are explained in greater detail at the end of the article.

Table 9 shows the versions of each language that were used during Kouatchou's research with the year of the survey highlighted. Least two rows show versions of languages used by the author of the work during repeating tests.

The published results are inconclusive and depend on the problem type. Some of the results presented are ambiguous presenting empty or partially empty results themselves making it impossible to indicate which Language performed better during these comparisons. These studies are presented in the last three rows of Table 8.

The algorithms showing the greatest advantage for Julia are listed in the first four rows of Table 8. Two of these namely the Laplace equation solver and matrix-copy operations demonstrated several fold performance gains. The other two achieved up to a 400 fold in which Julia outperformed Python.

Tab. 9. Language versions used by Kouatchou

Language	**Version**
Python 2019	3.7
Julia 2019	0.6.2
Python 2020	3.7
Julia 2020	1.2
Python 2021	3.9
Julia 2021	1.6.2
Python Repeated	3.11.2
Julia Repeated	1.8.5

The last three algorithms whose results indicate Python's advantage are placed in the middle of Table 8. Python achieved the best results for the algorithm calculating the value of the trigonometric function, with execution times several better than Julia. In the other two cases the advantage was marginal, amounting to only a few to several percent.

The results observed for any given problem were repeated regardless of the year of issue. Differences in results between years reached only a few percent, always in favor of the same language. It follows that neither the input data nor the software version (shown in Table 9) had a significant impact on the result of the comparison.

Although the results presented above are not definitive, they nonetheless suggest an overall advantage for the Julia language.

Similar conclusions can be drawn from studies conducted earlier in 2017, 2018 and 2019 and published by the author of the original research [17], [18], [19].

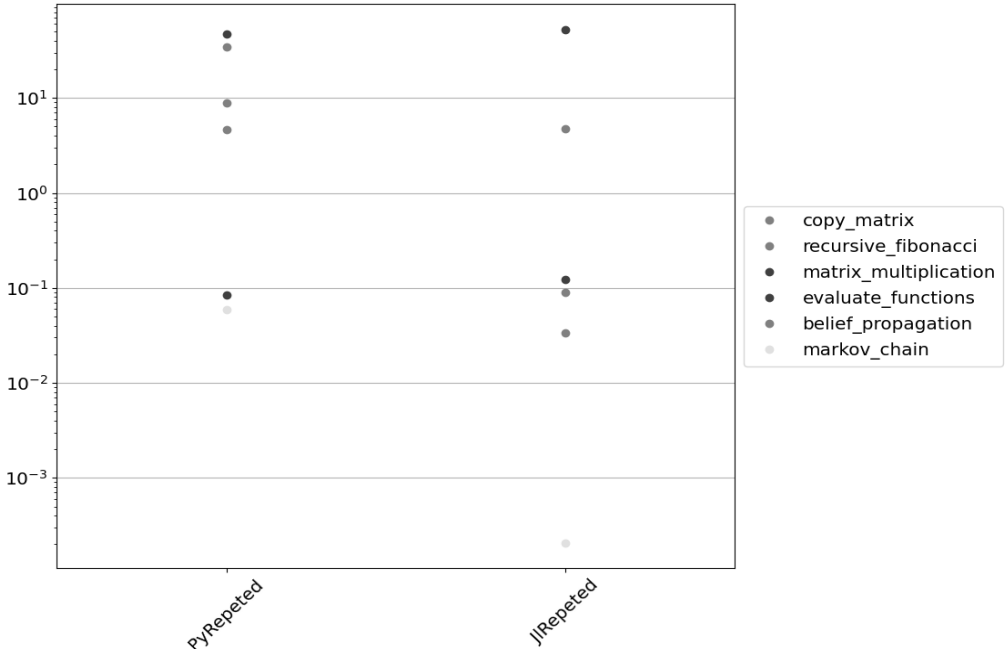

Fig. 6. Results of repeated tests presented by Kouatchou

Table 10 and Figure 6 show the results of the repeated tests. During the research were used the algorithms provided by Kouatchou [16]. Study were performed in the testing environment described in detail in the Table 1. Additionally used versions of Julia and Python can be find in Table 9 at least two rows.

Repeated tests were conducted on several selected algorithms. Some of them were discarded due to the need for a Linux environment, while others were selected due to slight differences in results, usually in favor of the Python language.

The results obtained in the repeated tests clearly confirm the correctness of the previously described results. The algorithms behaved in exactly the same way as the original tests. Operations such as copying matrices, calculating the n-th term of a Fibonacci sequence, and simulating a Markov process clearly speak in favor of Julia, achieving results more than 400 times better compared with Python.

Python shows an advantage in computing trigonometric functions, belief-propagation and matrix multiplication. However, as in the original research, the differences in favor of Python reach at most a dozen percent.

Tab. 10. Summary of results of repeated tests presented by Kouatchou

	Python Repeated [s]	**Julia Repeated [s]**
copy_matrix	34,63020	0,09034
recursive_fibonacci	8,87367	0,03381
markov_chain	0,05847	0,00021
evaluate_functions	47,32060	52,46225
matrix_multiplication	0,08377	0,12371
belief_propagition	4,62350	4,69460

3. Summary

3.1. Summary of research results

The test results posted on the Julia language's homepage [14], [15] clears that the Julia language performed all the given problems faster than Python.

The same conclusion is reached based on repeated tests in the research environment described in Table 1 performed on the basis of the algorithms provided on Julia's home page by the authors of the original test.

The advantage of the Julia language is clear from Selles' research [11]. With normalizing the results relative to C language, it can be seen that as the number of runs increases, Julia's results approach those of C. Python's performance, regardless of the number of calls, still remains about thirty times slower than C. With just one call, Julia is eleven times slower than C (three times faster than Python). Test on a hundred calls, the difference shrinks to twice as much in favor of C, while Python still remains thirty times slower than C, or fifteen times slower compared to Julia.

Lin's analysis [12], [13] shows that in most of the tests conducted, the package written in Julia achieved results better or comparable to packages developed in Python or using its interfaces. Julia, regardless of the type of input data, coped hopelessly with the problem of loading data. At the same time, it is worth noting that the packages in question may differ significantly in coding style and implementation method, which has a big impact on the tests results.

White's research [5] shows that Julia performs the given problems in only a few percent of the time it takes the Python language to perform the same problem. This score is the same for all the tests performed.

The results from repeated tests in the environment described in Table 1 by the author of the paper based on the algorithms provided by White allow different conclusions. Invariably, the Julia language outperformed Python in every test. Compared to the results achieved by the Python language, however, it performs at most twice as well when taking into considering results of tests solving the same problem.

Tests presented by Kouatchou [16], [20] produced mixed results. Some tests favored the Python interpreter, while others benefited from Julia compiler. Some results did not provide comparative data as it was described. Overall, Julia outperformed Python fully dominating in four algorithms evaluations. Whereas in three algorithms tests where Python held the edge, its advantage did not exceed a dozen percent.

The test repeated by the author of the work based on Kouatchou's [16] algorithms led to the same conclusions as the original tests. For each algorithm, Julia and Python exhibited virtually identical behavior, matching the performance differences reported in the initial study.

It should be noted that gross errors cannot occur in the repeated trials (by the author of the works with environment described in Table 1) due to the fact that the results from repeated tests are posted as an average of 10 independent algorithm executions, with any significantly deviant outcomes excluded.

In summary, in most existing publications, the Julia language shows significantly better performance than Python. However, everything depends on the way it is implemented and the types of libraries of each tool used.

3.2. Conclusion and direction of further research

Based on the analysis of existing literature and own research (as repeated test), it can be concluded that Julia performs assigned tasks significantly faster than Python.

There is little literature comparing the capabilities of Julia and Python languages. Therefore, it is necessary to carry out an analysis based on well-known algorithms widely spread in the programming world. Most importantly algorithms should be prepared based on pseudocode, with careful attention to maintaining uniformity in programming style and architecture (as suggested by White's guidelines) [5]. This will ensure that the research conducted will be authoritative what's more, the results will be unambiguous as to the results.

Further research should focus on comparing more complex operations. It would be interesting to see how GPU usage affects the execution times of each language. A good approach would also be to compare the languages on the CPU along the lines described earlier. What can be observed in the second part of the comparative analysis that this article deals with [21]. From the other hand it should be check how much influence the operating system has on the optimal execution of each algorithm.

The author of the paper continued the analysis on his own algorithms what is presented as another article and is a consistent reference to this article. The author of the work states that it should be considered as a coherent whole [21].

List of commonly used abbreviations and terms

.jl	Extension of programs written in the Julia language.
.py	Extension of programs written in Python language.
Julia	Julia programming language reference.
Python	Python programming language reference.
Algorithm	It is a set of instructions or logical rules specifying a sequence of operations to be performed by a computer system to solve a specific problem.
Execution time	The time required for a computer system or program to perform a specific task or operation. Execution time is measured in various units.
Development environment	A set of tools, libraries and other resources that are required to run and execute a computer program.
Editor	A computer program used to create, edit, and format text. In the context of computer science, a text editor is used to write source code for computer programs.
Compiler	It is a computer program that converts source code written in a programming language into equivalent machine code or intermediate code executable by the computer environment. This process is called compilation.
CPU	Central Processing Unit is the main processor of a computer, responsible for executing general-purpose tasks, managing system operations, and handling sequential processing efficiently.
GPU	Graphics Processing Unit is a specialized processor designed to handle parallel computations, primarily for rendering graphics and accelerating tasks like machine learning and scientific simulations.
RAM	Random access memory type of computer memory used to store working data and machine code.
copy_matrix	Function duplicates all elements from one three-dimensional array of size N x N x 3 to another, preserving both structure and order. It's primarily used to clone color matrices (e.g., RGB images) without altering their contents.
look_and_say	Algorithm generates successive terms of the "look-and-say" sequence, where each element verbally describes the counts and values of runs of identical digits in its predecessor.
iterative_fibonacci / recursive_fibonacci	Algorithms that compute the n-th Fibonacci number, differing only in technique one uses an explicit loop, the other uses recursive calls.
evaluate_functions	Algorithm that computes one or more trigonometric functions at N evenly spaced sample points over a given interval (for example, [0, 2π]).
belief_propagation	Algorithm performs a specified number of iterations of belief message passing on a probabilistic graphical model to estimate marginal distributions.
markov_chain	Function simulates a Markov process for a given number of steps, transitioning between states according to a transition probability matrix.
laplace_equation	Algorithm solves the two-dimensional Laplace equation on an N X N grid (commonly via Gauss–Seidel relaxation).
munchauser_number	Function searches for Münchausen numbers in the range 0 to N, where each candidate equals the sum of the factorials of its digits.
pernicious_number	Algorithm finds the n-th pernicious number integers whose binary representations contain a prime number of ones.
iteration_pi_sum	Algorithm that calculates the approximate value of the number π (pi) by means of an infinite series sum.
matrix_multiply / matrix_multiplication	Algorithm that calculates the value of multiplying two matrices.
matrix_statistics	Algorithm that calculates statistics for a given matrix, such as mean, standard deviation, minimum and maximum value, etc.
parse_integers	Algorithm that analyses a string to extract integers from text.
print_to_file	Algorithm for outputting data or calculation results to a file.
recursion_quicksort	Fast-sorting algorithm that uses a recursive approach.
recursion_fibonacci	Algorithm for calculating the Fibonacci number for a given index using recursion.
userfunc_mandelbrot	Algorithm that visualizes the Mandelbrot set.
n-body problem	Problem in physics and computer simulation that involves predicting their motion (especially objects that interact with each other by gravitational or electrostatic forces).
Binary tree	Binary tree is a dynamic data structure that consists of nodes connected by edges in such a way that each node has a maximum of two left and right descendants.

Shortest path problem	Shortest path problem – a problem in graph theory involving finding the shortest path from one node to another arbitrarily chosen node in a graph.
K-core decomposition	K-core decomposition is a graph analysis technique that divides a graph into groups based on degree order and graph topology. That is, maximal subgraphs in which each node has at least degree k.
Strongly Connected Components	Groups of vertices in a graph in which every vertex can be reached from every other vertex in the same component by a directed path.
PageRank	An algorithm developed by Larry Page and Sergey Brin that has been used to assess the relevance of web pages (a method of determining the quality of indexed web pages by assigning numerical values to them). In the context of graphs and networks, nodes in a graph are given a numerical value based on the number of connections to other nodes.
Loading	The data problem involves loading data. The data being loaded comes in a variety of formats-for example, database text files or spreadsheets. The process may involve cleaning the data (adapting it for analysis) or converting it to another format.

4. Bibliography

[1] "General Python FAQ", 3.11.11 Documentation, Python Software Foundation (online: 04.03.2025).

[2] "Frequently Asked Questions", Julia Language Documentation, Franklin.org and others, (online: 04.03.2025).

[3] "History of Julia", Julia Wiki, Wikimedia Developer Portal (online: 30.10.2023).

[4] "Julia programming language", University of Cincinnati Libraries, 2023 (online: 04.03.2025).

[5] White M., „The Need for Speed: Julia vs. Python", 2022, DOI: 10.13140/RG.2.2.16468.32646.

[6] "Julia in Visual Studio Code", Microsoft, (online: 04.03.2025).

[7] "Python in Visual Studio Code", Microsoft, (online: 04.03.2025).

[8] "Learn To Code With Visual Studio Code", Microsoft (online: 04.03.2025).

[9] "Jupyter Notebooks in VS Code", Microsoft (online: 04.03.2025).

[10] "JuliaLang/ Microbenchmark", "GitHub", Franklin.org and others (online: 04.03.2025).

[11] Sells R., "Julia Programming Language Benchmark Using a Fight Simulation", *IEEE Conference on Aerospace*, MT, USA, 2020, DOI: 10.1109/AERO47225.2020.9172277.

[12] Lin T., "Benchmark of popular graph/network packages v2", Quasilinear Musings, 2020 (online: 04.03.2025).

[13] Lin T., "Benchmark of popular graph/network packages", Quasilinear Musings, 2019 (online: 04.03.2025).

[14] "Benchmarks", Franklin.org and others, (online: 04.03.2025).

[15] "Julia language home page", Franklin.org and others (online: 04.03.2025).

[16] Kouatchou J., "Basic Comparison of Various Computing Languages", GitHub (online: 15.05.2025).

[17] Kouatchou J., "Basic Comparison of Python, Julia, Matlab, IDL and Java (2017 Edition)", basic_language_comparison.

[18] Kouatchou J., "Basic Comparison of Python, Julia, Matlab, IDL and Java (2018 Edition)", basic_language_comparison.

[19] Kouatchou J., Medema A., "Basic Comparison of Python, Julia, Matlab, IDL and Java (2019 Edition)", basic_language_comparison.

[20] Kouatchou J., "Basic Comparison of High--Level Programming Languages", (online: 15.03.2025).

[21] Maj M., "Performance comparison of Julia and Python programming languages on GPU and CPU", *Computer Science and Mathematical Modelling*, No. 21, 5–17 (2024) will be published.

COMPUTER SCIENCE AND MATHEMATICAL MODELLING **20**: 5–17 (2024)

Porównanie wydajności języków oprogramowania Julia i Python na podstawie istniejącej literatury

M. MAJ

Język Julia jest szybko rozwijającym się językiem programowania wysokiego poziomu. Jego twórcy sugerują, że jakość i implementacja innowacyjnej technologii kompilacji (Just-In-Time Compilation) pokonuje Pythona pod względem czasu wykonywania algorytmów. Wybór języka programowania do implementacji algorytmu ma ogromne znaczenie. W dzisiejszym świecie każda zaoszczędzona milisekunda może zadecydować o sukcesie produktu. Wybór ma bezpośredni wpływ na wydajność aplikacji lub wykonanie algorytmu. W związku z tym analiza porównawcza nowoczesnych lub powstających technologii staje się szczególnie ważna. Artykuł ten stanowi pierwszą część analizy porównawczej, której kontynuacja [21] rozszerza badania dotyczące tematyki niniejszej pracy. W tym kontekście w artykule przedstawiono podsumowanie istniejącej literatury i omówiono wydajność czasową języków programowania Julia i Python. Ponadto wyniki testów i opublikowane kody źródłowe zostały odtworzone i zweryfikowane w środowisku badawczym. W dalszej części artykułu [21] badania zostały rozszerzone o kolejne algorytmy oraz o sprawdzenie wydajności każdego z języków na kartach graficznych.

Słowa kluczowe: Python, Julia, porównanie, wydajność.