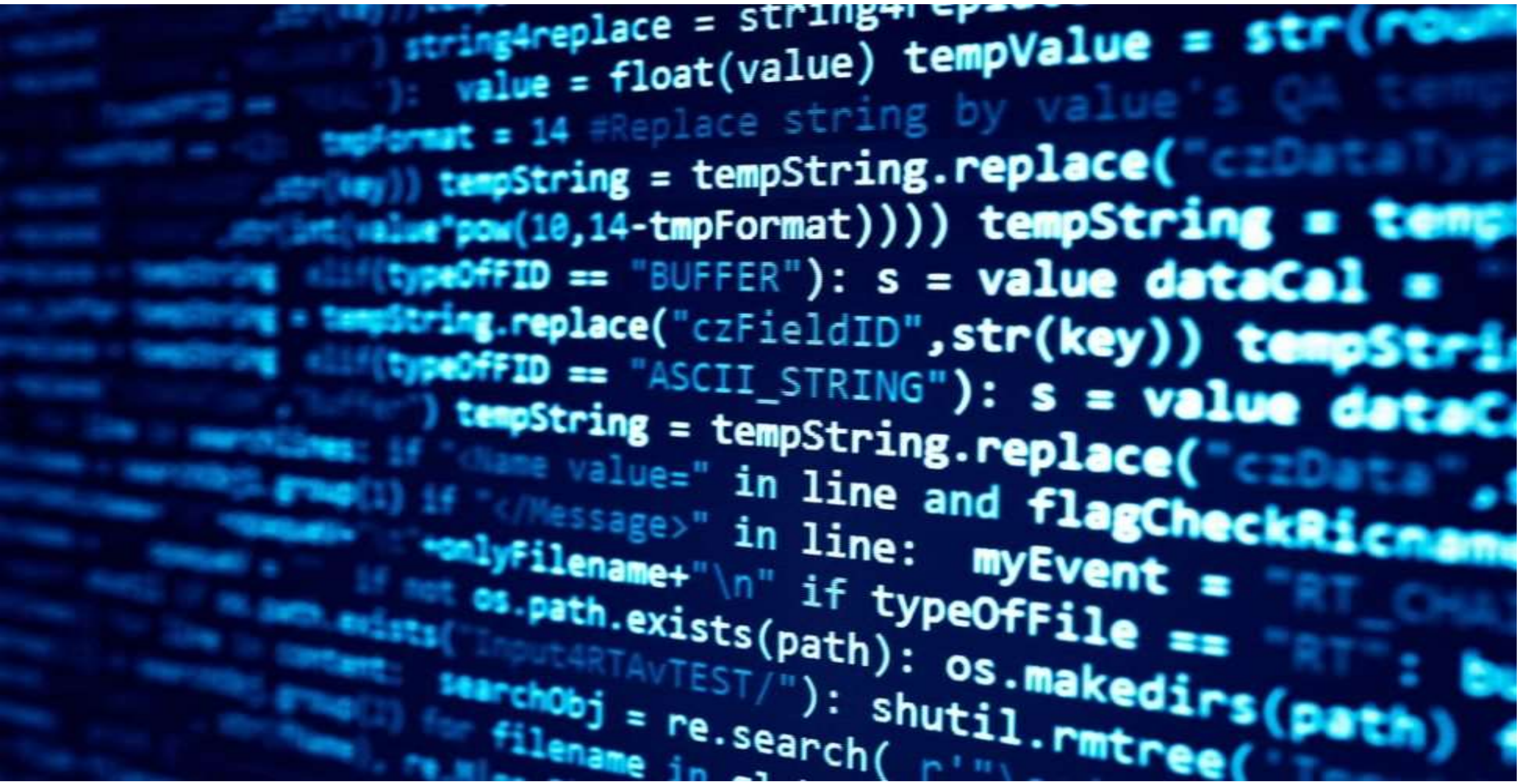
With the support of the Erasmus+ Programme of the European Union

School of Built Environment, Engineering, and Computing

ICT and Environment

Comparative Analysis of Energy Consumption in Python, Java, C, C++, and JavaScript

Student: Manasik Hassan, c7302653
Supervisor: Dr. Ah-Lian Kor



Abstract

The ICT industry is valuably supporting other industries in reducing their carbon footprint. However, ICT poses a real challenge as a substantial contributor to GHG emissions. Most significant efforts in greening the ICT are directed toward hardware components, and less consideration for the software carbon footprint was observed. In favor of this, and as a contribution to recommending energy-aware software practices; this work investigates the energy efficiency of the top five most popular programming languages according to the famous IEEESpectrum website (Python, Java, C, C++, and JavaScript) by comparing their energy consumption in four benchmarks presenting different CPU intensity levels and various computational tasks. By using Microsoft Joulemeter to measure the energy consumption and DEFRA carbon footprint conversion factors, our results showed that Java and C are more energy-efficient and energy stable regardless of task complexity than Python, JavaScript, and C++. Moreover, although there are cases where Python showed optimum energy performance, its overall energy consumption patterns are the least efficient and sensitive to task complexity.

Table of Contents

1 Introduction

With the adoption of the 2030 agenda for sustainable development by all the United Nations Members in 2015 to enhance human lives and protect the ecosystem (THE 17 GOALS | Sustainable Development), rising efforts have been directed to reduce GHG emissions from all various industries.

Although the ICT industry is valuably supporting achieving all the 17 SDGs (ITU, 2018)**,** many researchers claim that ICT is not receiving enough consideration as a substantial contributor to GHG emissions. Many available estimates indicate that ICT is responsible for 1.8-2.8% of the GHG emissions; however, (Freitag *et al.*, 2020) suggest that the actual contribution to the GHG emissions is higher by almost 25% than the available official estimates. Furthermore, with our growing dependence on ICT and the increasing no. of ICT services and equipments, this percentage could exceed 14% of the 2016-level global GHGE before 2040 (Belkhir and Elmeligi, 2018). In favor of the previous figures, Green ICT practices are becoming essential.

1.1 Problem Statement

ICT systems are composed of devices connected using various networking schemes. Each node in the ICT infrastructure consists of hardware, software, and an operating system that manages the hardware and software resources.

(Ardito and Morisio, 2014) and (Kumar, Li and Shi, 2017) highlighted that most Green ICT practices focus on the energy efficiency and energy optimizations for the hardware components, and efforts toward tracking and optimizing the energy performance in the software layer are still in their early stages. They also claimed that energy savings obtained from software components distributed all around the globe could be more than energy savings from hardware; yet, this requires innovative tools and approaches to assess the energy consumption of different elements in ICT.

Conventionally, software programmers receive intensive education about the programming languages and the methods needed to develop highly performed applications. The energy-efficiency aspect of these programming approaches isn't included in the education. Hence, they don't think about the energy performance of their software solutions, and most of their attempts to enhance the software performance focus solely on the execution time (Kumar, Li and Shi, 2017). Although run time optimization is often related to energy optimization, it is an inadequate measure to depend on for greener software (Couto *et al.*, 2017), (Pereira *et al.*, 2016), (Pang *et al.*, 2016). A survey was conducted in 2016 on more than 100 programmers to review their awareness of energy-efficient software best practices and how software implementations affect energy consumption in devices. The survey results showed that most of them lack the basic knowledge to address this problem (Pang *et al.*, 2016).

This situation demands more efforts to develop energy-aware best practices guidelines for software development and the programming languages in use.

1.2 Scope Of the Research

Many programming languages and software development frameworks exist. The Scope of this research is limited to studying and comparing the energy efficiency of the top five most popular programming languages used in software development according to the IEEESpectrum website: (Python, Java, C, C++, and JavaScript). The assessment is done by comparing the energy consumption of these languages when solving four different computational problems. The motivation of this research is to contribute to the energy-aware software development field and support software developers in understanding the energy cost of the available alternatives.

1.3 Aims And Objectives

1.3.1 Research Aims

This research aims to deliver an in-depth understanding of methodologies and tools used to assess energy consumption and GHG emission from the ICT industry. Therefore, focusing on five well-known programming languages and applying these methodologies to compare their energy performance.

1.3.2 Research Objectives

To investigate and understand the energy consumption traits of the studied programming languages when used to solve various computational problems and answer the following research questions:

R.Q 1: Is there a significant difference in the energy consumption of Python, Java, JavaScript, C, and C++?

R.Q 2: What is the ranking of these programming languages energy-wise?

R.Q 3: Is the fastest programming language the greenest as well?

R.Q 4: Is the most popular programming language recently (Python) energy efficient compared to other programming languages on the list?

1.4 Outline Of the Report

The rest of this report is organized as follows: In Literature Review, previous work related to comparing energy consumption between multiple software languages is shown. The Methodology section will demonstrate various macro and micro methodologies conducted for this research, followed by the Results and Discussion section to interpret our findings, and then Recommendations will be given. Finally, we conclude our work and highlight intended future studies in the Conclusion and Future Work Section.

2 Literature Review

This section introduces the related research work found in the literature review, followed by a brief description of our study's programming languages.

2.1 Energy Consumption Evaluation for Programming Languages

Investigating the Energy efficiency of the programming languages started to gain proper attention only in recent years. This subsection shows some of the related work found in the literature.

Many researchers focus on the energy efficiency of programming languages used in developing mobile applications as this directly affects the battery performance of handheld devices. For instance, (Oliveira, Oliveira and Castor, 2017) compared the energy consumption of using Java, JavaScript, and C/C++ programming languages in developing Android Applications and investigated the enhancement in energy efficiency when using a combination of two different programming languages instead of writing the whole app in the same language. Their findings highlighted that even though Java and C++ performance showed to be better than JavaScript, the latter was more energy efficient in most of their 33 benchmark tests. Moreover, they indicated that developing mobile applications with hybrid languages can sometimes improve the application's energy consumption. (Rashid, Ardito and Torchiano, 2015) carried out a study to review the energy performance of ARM assembly language, Java, and C in mobile applications. They executed various sorting algorithms to conclude that ARM assembly is the greenest to solve sorting problems among the three languages. (Corral *et al.*, 2014) carried out multiple experiments to investigate whether it's possible to indicate the energy consumption of a piece of code from its execution time and which programming language between C and Java is more convenient. Their work showed that C outperformed Java in both time and energy contexts in all their benchmark problems. Additionally, (Kholmatova, 2020) conducted a meta-analysis to assess the energy performance of C, C++, and Java in writing mobile applications. Their analysis concluded no significant difference in the energy consumption for portable applications written in C, C++, or Java.

Away from the mobile application domain, multiple research works aimed to compare the energy efficiency of programming languages in a general context. To start with, (Hamizi et al., 2021) conducted a meta-analysis to compare the energy consumption of Python and Java. Their results showed no remarkable difference in the energy performance between these two programming languages. (Noureddine et al., 2012) used POWER API OS library to measure the energy consumption of several Towers of Hanoi implementations for multiple programming languages. Their results revealed that Java is the most energy-efficient language, followed by C and C++, while Perl is the least energy efficient. Moreover, (Couto et al., 2017) work aimed to create a performance ranking list for ten programming languages (C, C#, Fortran, Go, Java, JRuby, Lua, OCaml, Perl, and Racket) in terms of energy consumption, execution time and memory usage. They used different benchmark programs for solving computational problems written in these ten languages. Their primary finding is that C is the greenest language on the list, and another conclusion is that compiled languages outperformed interpreted ones in terms of speed and energy savings. However, they highlighted that the famous claim that "the faster the program, the greener it is" is not always valid. The same authors later in 2017 extended their experiments to include around 27 programming languages using the same experimental setup mentioned earlier. This extended ranking work concluded that C is the fastest and greenest programming language, while Pascal is the language with the least memory usage (Pereira *et al.*, 2017).

It’s plain to see that there are no sufficient conclusions and unified ranking for the energy performance of programming languages. Hence this research aims to contribute to the existing studies and help understand the energy traits for the most popular programming languages introduced in the next section.

2.2 Programming Languages

2.2.1 Python

Python is an interpreted, general-purpose, high-level programming language firstly introduced in 1991 by Guido van Rossum as a descendant of the ABC language. It’s a multi-paradigm language that supports several programming styles, such as object-oriented and structured programming. Python is a dynamic language that executes many tasks during runtime rather than compile time. Python developers’ philosophy tries to avoid building patches that enhance the speed at the cost of clarity and readability, and hence for many solutions, it’s slower than many other languages. Instead of implementing all python functions within its core, python developers designed it to be highly extensible via modules, and a programmer can use extensions modules written in C to implement the time-sensitive code parts, for example (Wikizero - Python (programming language), no date), (van Rossum, 1995).

Python learning is quite unchallenging, and learners become productive quickly, the reason why it continues to dominate as the most popular programming framework for various ICT domains (Li, 2016). However, there are many limitations in its performance, and thus it’s not suitable to be used for multiple domains (Top Programming Languages 2021, no date).

2.2.2 C

The C programming language was initially developed in the early 1970s by Dennis Ritchie and Brian Kernighan at Bell Labs as a system language for Unix operating system. They were inspired by BCPL, a type-less language (Wikizero - The C Programming Language, no date), (Chistory, no date).

C is a general-purpose, procedural, structured, abstract programming language that proved efficient in system applications and replaced the previously used assembly languages. Even though it was not initially developed for small-scale portable machines, the use of the C language witnessed a huge spread, and C compilers became everywhere (Chistory, no date). C can be used to integrate every programming approach in almost all ICT areas, it is efficient when it comes to the use of hardware resources, and most significant, C compliers with full support for C language are almost available in every computer/machine regardless of its capabilities (Stroustrup, 1986)

2.2.3 C++

C++, initially called C with classes, was designed by Bjarne Stroustrup at AT&T Bell Labs to become an extension and a better version of the C programming language. It became available for commercial use in 1985, and the goal of its developers was to bring out a programming language that is suitable for a wide variety of programming styles in highly demanding real-world applications (Stroustrup, 1999).

C++ is a general-purpose, multi-paradigm programming language that involves object-oriented, generic, and functional programming styles. It was designed to integrate both the organizational and design strength of Simula language(object-oriented) with the system programmability and abstraction options provided by C. Hence, C++ considers C a subset language. The key differences between C and C++ is that language constructs are allowed only in C++, and multiple features were added to C++ to avoid the mostly occurred errors in C (Wallace, 1993), (Stroustrup, 1986), (Stroustrup, 1999).

Although it was initially developed for systems implementations, its capabilities proved useful for many other technical domains; it found its way to rank fourth among the most popular programming languages in 2022, after Python, Java, and C (*Wikizero - C++*, no date).

2.2.4 Java

Java was firstly introduced in 1995 by James Gosling at Sun Microsystems, initially intended for small applications such as interactive television and portable devices. It's now one of the most popular programming languages running almost everywhere in the ICT industry, and it's remarkably used for backend web developments (Dmitriev, 2002), (*Wikizero - Java (programming language)*, no date).

Java is a classed-based, object-oriented programming language whose structure is based on objects and classes and not only functions. It's a high-level language in which the language doesn't provide a detailed representation of the JVM machine. It was typically developed to have as few implementation dependencies as possible. Java written code is usually compiled to a bytecode format and can be executed in any JVM without recompiling the code. Java Syntax is close to C and C++ syntax but is a less complex style. Finally, although java is a static language, its runtime environment offers dynamic capabilities like reflection and runtime code modifications which are usually not available in other compiled programming languages (Gosling et al., 2005), (Wikizero - Java (programming language), no date), (Van Hoff, 1997).

2.2.5 JavaScript

JavaScript was first introduced in 1996 by Eich at Netscape to overcome the problem of static web page design. It is designed with a close syntax to java to become a simple scripting language and support dynamic customization of the web pages (Wirfs-Brock and Eich, 2020), (*Wikizero - JavaScript*, no date).

JavaScript is a high-level, multi-paradigm programming language that makes it possible for dynamic typing, object orientation, just-in-time compilation, and functional and imperative programming techniques to be adopted by its user (*Wikizero - JavaScript*, no date), (Kienle, 2010). Such as Java, JavaScript derived its syntax and structs from the C language (Mikkonen *et al.*, 2007), (Severance, 2012). Although it is considered the de facto for the client-side web programming, the development of multiple JavaScript core runtime systems helped in using JS to build software applications previously written only in C, C++, and Java (*Wikizero - JavaScript*, no date), (Mikkonen *et al.*, 2007).

3 Methodology

3.1 Macro level Methodology

With the increasing attention to assessing the environmental impact of the ICT, many standardized methodologies were introduced to define an overall approach that guides ICT practitioners on how to conduct Life cycle assessments for their products and services.

After considering the available options, the ITU-T L.1410 Life cycle assessment framework will be adopted to achieve the objectives of this research. This framework was released by International Telecommunication Union in 2014 to aggregate previous standards (ISO 14040 and ISO 14044) and to provide a more detailed guideline (ITU-T, 2014). It covers all components in the ICT, e.g., goods, networks, and services in all Life cycle stages, and hence it defines broad steps for LCA applications. Therefore, with relevance to our scope of assessing an ICT service in the usage stage, a customized version of ITU-T L.1419 with five phases will be applied in this report.

These five phases are described as follows:

1- **Goal and Scope Definition:** This phase is associated with identifying the targeted ICT components to be assessed, presenting aims and objectives for the analysis, and setting the research boundaries.
2- **Life cycle Inventory (LCI):** LCI is related to all data collection and data processing methods applied in the research.
3- **Life cycle Impact Assessment (LCIA):** In this step, results from the LCI phase are projected into environmental-related measures. In this work, CO_2 emissions will be used as an LCIA indicator.
4- **Life Cycle Interpretation:** In the interpretation phase, results from LCI and LCIA steps are discussed and evaluated according to the proposed aims and objectives in the first phase. Limitations, shortcomings, and uncertainties are highlighted in this phase as well.
5- **Reporting:** in the last stage, the whole LCA process will be documented and summarized, and recommendations and suggestions will be provided.

The following figure illustrates the adopted methodology.

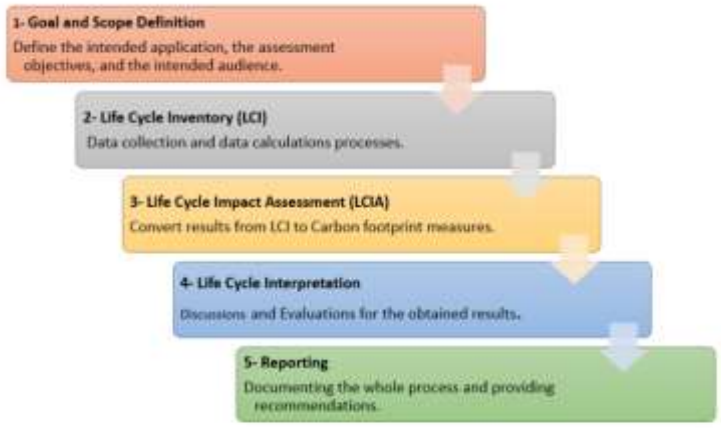

1- Goal and Scope Definition
Define the intended application, the assessment objectives, and the intended audience.
2- Life Cycle Inventory (LCI)
Data collection and data calculations processes.
3- Life Cycle Impact Assessment (LCIA)
Convert results from LCI to Carbon footprint measures.
4- Life Cycle Interpretation
Discussions and Evaluations for the obtained results.
5- Reporting
Documenting the whole process and providing recommendations.

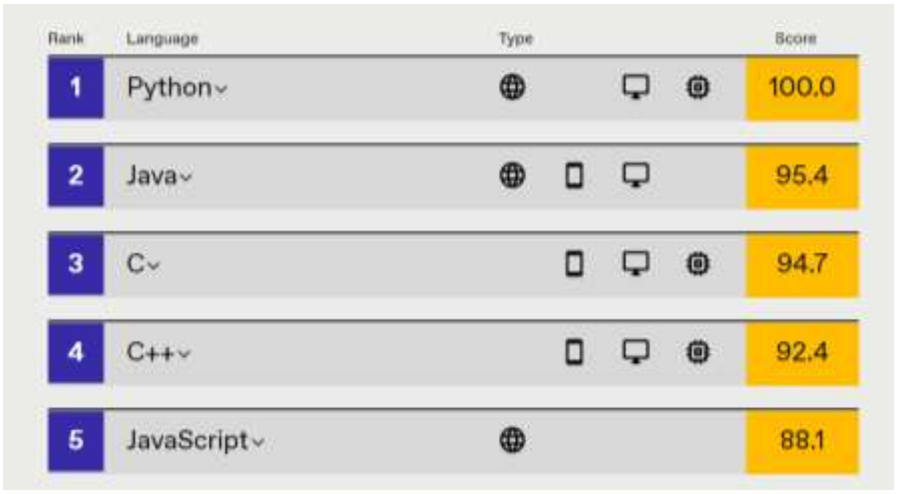
Figure 1. ITU-T L.1410 LCA Framework (ITU-T, 2014).

3.2 Micro Level Methodology

3.2.1 Programming Languages Selection Procedure

Countless Programming languages and software frameworks exist; however, this research is limited to studying the five most popular programming languages in recent years. Multiple ranking platforms were investigated, and IEEESpectrum 2021 ranking was chosen as the languages selection baseline. IEEESpectrum ranking website uses popularity indicators of programming languages from eight famous platforms associated with software development (CareerBuilder, GitHub, Google, Hacker News, the IEEE, Reddit, Stack Overflow, and Twitter).

The website offers rankings based on multiple criteria, such as the language domain of use (web, Enterprise, Mobile, and Embedded), the most trending, or the languages with the most job openings. The default ranking setup was chosen for this research, which ranks the language's popularity across all development sectors. The following figure shows the top five popular programming languages and their scores and domain of use (*Top Programming Languages 2021*, no date).



Rank	Language	Type	Score
1	Python	🌐 🖥 ▣	100.0
2	Java	🌐 📱 🖥	95.4
3	C	📱 🖥 ▣	94.7
4	C++	📱 🖥 ▣	92.4
5	JavaScript	🌐	88.1

Figure 2. Programming languages popularity ranking in 2021 according to IEEE Spectrum website.

3.2.2 The Benchmark Problems

To obtain an accurate and broad comparison of the energy performance of the previously mentioned languages, it's essential to propose different programming algorithms to be executed.

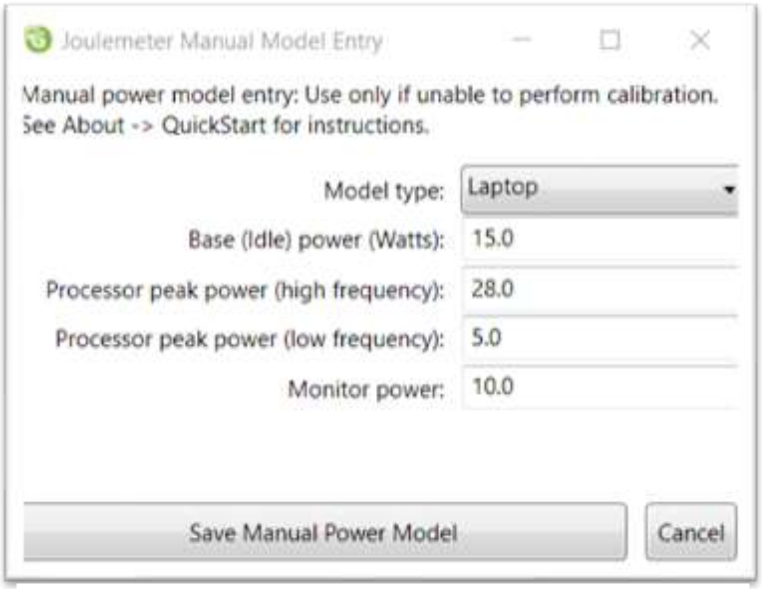
Four distinct programs for different CPU stressing levels were executed. We proposed the first two programs presenting low CPU utilization. However, the remaining two programs were obtained from the famous TCLBG website, which are CPU-intensive computational problems to stress the CPU to understand energy behaviors better. The Computer Language Benchmark Game is a project to compare the performance of different programming languages in terms of runtime and memory utilization. Various computational problems are proposed on the TCLBG website, and these problems are solved in more than 25 languages using only the same algorithm. For each case, there is a predefined set of steps proposed on the website, and participating software programmers for all languages have to follow this algorithm when writing their codes to solve the specific problem (Gouy, 2008).

In our experiments, codes written in Python, Java, C, C++, and JavaScript for all four benchmarks were executed. Following is a description of the adopted benchmarks that are available in the appendix section:

1- An iteration program to print “Hello World” one million times using for loop.
2- A Recursion program to add numbers from 1 to 100. The recursion task is iterated one million times.
3- N-body Program from TCLBG: This program performs mathematical calculations to predict the position and velocity of N-body in a group of objects interacting with each other’s gravitationally using given execution instructions on the website.
4- Fannkuch-redux Program from TCLBG: This program reverses and reorders a sequence of numbers in the range from 1 to 12, accessing their indices, and then outputs the number of flips needed to make the first element in the sequence 1.

3.2.3 Energy Profiler Tool

Microsoft Joulemeter was used to estimate the power consumption when running each of our codes. It tracks the energy consumption by converting the PC resource utilization to actual power measurements based on the learned power model for the PC during the calibration process (*Microsoft Joulemeter*). The following figure shows the manual calibration setup for the joulemeter.



Manual power model entry: Use only if unable to perform calibration. See About -> QuickStart for instructions.

Model type: Laptop
Base (Idle) power (Watts): 15.0
Processor peak power (high frequency): 28.0
Processor peak power (low frequency): 5.0
Monitor power: 10.0

Figure 3. Joulemeter manual Calibration parameters

3.2.4 Experimental Setup

Lenovo Yoga Laptop was used as a testbed to conduct our experiments. The following table gives the specification for the laptop.

Table 1. Technical Specifications of the PC.

Features	Specifications
Model	YOGA C640-13IML
Operating System	Windows 10 Home, 64-bit Operating System
Processor	Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
GPU	Integrated Intel® HD Graphics 620
Screen	13.3" FHD (1920 x 1080) pixels
Hard Disk	512 GB SSD
RAM	16 GB

Command Line Interface was used to execute the codes for each of our four cases mentioned in the previous section. All other background tasks and running applications except CMD and Joulemeter were terminated before starting the experiment to maintain consistency. Each of our twenty source codes (5 programming languages * 4 algorithms) was iterated ten times to verify the results and avoid possible outliers. The experiment was conducted with the programming release shown in the following table:

Table 2. Programming Languages Releases.

Programming Language	Release
Python	v3.9.7
Java	OpenJDK 11.0.13 2021-10-19
C	gcc (MinGW.org GCC-6.3.0-1) 6.3.0
C++	g++ (MinGW.org GCC-6.3.0-1) 6.3.0
JavaScript	Node.js v16.9.1

4 Results and Discussion

This section shows the results of conducting 200 experiments for our five studied programming languages across four different scenarios to answer the research questions proposed in section 1.3.2. 1.3.2

4.1 One-way ANOVA test

To investigate whether there is a significant difference in energy consumption levels between C, C++, Java, JavaScript, and Python in different benchmarks (Iteration Program, Recursion Program, N-body program, and Fannkuch-redux program); a One-way ANOVA test was performed assuming the following hypothesis:

- **Null Hypothesis (H_0):** There is no remarkable difference in the mean values for energy consumption levels between C, C++, Java, JavaScript, and Python.
- **Alternative Hypothesis (H_a)**: At least one of the means is significantly different from the others.

Working with the one-tail ANOVA test and confidence level of (95%), alpha value, $\alpha = 0.05$, the following results were obtained. The source code for ANOVA testing is available in the appendix.

```
One-way ANOVA testing for fannkuch-redux Benchmark
One-way ANOVA testing for CPU (W)
F Value=233.621, P Value=1.78397e-29   The null hypothesis can be rejected

One-way ANOVA testing for Monitor (W)
F Value=98.1865, P Value=1.24696e-21   The null hypothesis can be rejected

One-way ANOVA testing for Disk (W)
F Value=4.95165, P Value=0.00214876    The null hypothesis can be rejected

One-way ANOVA testing for Base (W)
F Value=97.5624, P Value=1.4173e-21    The null hypothesis can be rejected

One-way ANOVA testing for Total Hardware Power (W)
F Value=144.294, P Value=4.72196e-25   The null hypothesis can be rejected

One-way ANOVA testing for Application (W)
F Value=209.505, P Value=1.82933e-28   The null hypothesis can be rejected
```

Figure 4. One-way ANOVA testing for fannkuch-redux Benchmark

For Fannkuch-redux Benchmark, the hardware and application energy consumption levels vary significan tly between Python, Java, JavaScript, C, and C++ languages.

One-way ANOVA testing for N-body Benchmark

```
One-way ANOVA testing for CPU (W)
F Value=23516.8, P Value=4.58258e-74   The null hypothesis can be rejected

One-way ANOVA testing for Monitor (W)
F Value=76797.7, P Value=1.25941e-85   The null hypothesis can be rejected

One-way ANOVA testing for Disk (W)
F Value=5.91803, P Value=0.000648675   The null hypothesis can be rejected

One-way ANOVA testing for Base (W)
F Value=152400, P Value=2.53556e-92    The null hypothesis can be rejected
One-way ANOVA testing for Total Hardware Power (W)
F Value=129828, P Value=9.34039e-91    The null hypothesis can be rejected

One-way ANOVA testing for Application (W)
F Value=23764.3, P Value=3.62143e-74   The null hypothesis can be rejected
```

Figure 5. One-way ANOVA testing for N-body Benchmark

The hardware and application energy consumption levels for N-body Benchmark significantly differ betw een Python, Java, JavaScript, C, and C++ languages.

One-way ANOVA testing for Iteration Benchmark

```
One-way ANOVA testing for CPU (W)
F Value=25.303, P Value=5.06534e-11    The null hypothesis can be rejected

One-way ANOVA testing for Monitor (W)
F Value=1157.83, P Value=9.80603e-45   The null hypothesis can be rejected

One-way ANOVA testing for Disk (W)
F Value=1.30426, P Value=0.28285       The null hypothesis is accepted

One-way ANOVA testing for Base (W)
F Value=1157.83, P Value=9.80603e-45   The null hypothesis can be rejected


One-way ANOVA testing for Total Hardware Power (W)
F Value=391.146, P Value=2.5439e-34    The null hypothesis can be rejected


One-way ANOVA testing for Application (W)
F Value=253.95, P Value=2.97578e-30    The null hypothesis can be rejected
```

Figure 6. One-way ANOVA testing for Iteration Benchmark

For Iteration Benchmark, the overall hardware and application energy consumption levels are significantl y different between Python, Java, JavaScript, C, and C++ languages. However, our Null Hypothesis is only valid for Disk Energy performance in this benchmark.

```
One-way ANOVA testing for Recursion Benchmark
One-way ANOVA testing for CPU (W)
F Value=412.94, P Value=7.77544e-35    The null hypothesis can be rejected

One-way ANOVA testing for Monitor (W)
F Value=3840.31, P Value=2.20782e-56   The null hypothesis can be rejected

One-way ANOVA testing for Disk (W)
F Value=0.971884, P Value=0.432317     The null hypothesis is accepted

One-way ANOVA testing for Base (W)
F Value=3840.31, P Value=2.20782e-56   The null hypothesis can be rejected

One-way ANOVA testing for Total Hardware Power (W)
F Value=2980.26, P Value=6.4996e-54    The null hypothesis can be rejected

One-way ANOVA testing for Application (W)
F Value=784.013, P Value=5.68313e-41   The null hypothesis can be rejected
```
Figure 7. One-way ANOVA testing for Recursion Benchmark

For Recursion Benchmark, the total hardware and application energy consumptions significantly differ be tween Python, Java, JavaScript, C, and C++ languages. Nevertheless, our Null Hypothesis is only valid for Disk Energy performance in this benchmark.

The following Table Summarizes the p-values results for One-way ANOVA testing.

Table 3. One-way ANOVA testing results

P-Value for One-way ANOVA testing	CPU (W)	Monitor (W)	Disk (W)	Base (W)	Total Hardware Power (W)	Application (W)
Fannkuch-Redux Program	1.78397E-29*	1.24696E-21*	0.00214876*	1.4173E-21*	4.72196E-25*	1.82933E-28*
N-body Program	4.58258E-74*	1.25941E-85*	0.000648675	2.53556E-92*	9.34039E-91*	3.62143E-74*
Hello Program	5.06534E-11*	9.80603E-45*	0.28285	9.80603E-45*	2.5439E-34*	2.97578E-30*
Recursion Program	7.77544E-35*	2.20782E-56*	0.432317	2.20782E-56*	6.4996E-54*	5.68313E-41*

The asterisk (*) indicates this P-value rejects the Null Hypothesis for the confidence level of (95%) and alpha value, $\alpha = 0.05$

The following points can summarize the One-way ANOVA test results:

- Using inferential statistics tests, it's apparent that there is a significant difference in the energy consumption levels of C, C++, Java, JavaScript, and Python. In particular, total hardware and application power are significantly different no matter what instructions are executed (Iteration Program, Recursion Program, N-body program, and Fannkuch-redux program).
- Detailed explanations will be discussed in the following subsections.

4.2 Energy Consumption of Programming Languages

The following subsections discuss the energy behavior of C, C++, Java, JavaScript, and Python for each benchmarking.

4.2.1 Results of Iteration Benchmark

The iterative hello world program presents our least CPU-intensive task. The objective was to test which programming language is energy-efficient in the commonly implemented iteration process. The following table and figure illustrate the energy consumption results across the different hardware and software components.

Table 4. Energy Consumption of C, C++, Java, JavaScript, and Python with Iteration.

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Aggregated Monitor (J)	Aggregated Disk (J)	Aggregated Base (J)	Aggregated Hardware Energy (J)	Aggregated Application (J)	Aggregated Total Energy Consumption (J)	Total Time to execute the task (s)	Average Power Consumption (Watt/sec)
C	81.75876	201.1585	0	502.8963	785.8136	23.02502	808.8386	34.019	23.77608
C++	130.7004	402.0878	0.010162	1005.22	1538.018	46.33757	1584.356	68.0425	23.28479
Java	88.36957	212.0374	0.010478	530.0935	830.511	26.7868	857.2977	35.8268	23.92895
JavaScript	140.277	243.0158	12.10411	607.5396	1002.937	39.75343	1042.69	41.0216	25.41807
Python	66.4872	143.5702	0	358.9254	568.9828	22.3341	591.3169	24.2336	24.4007

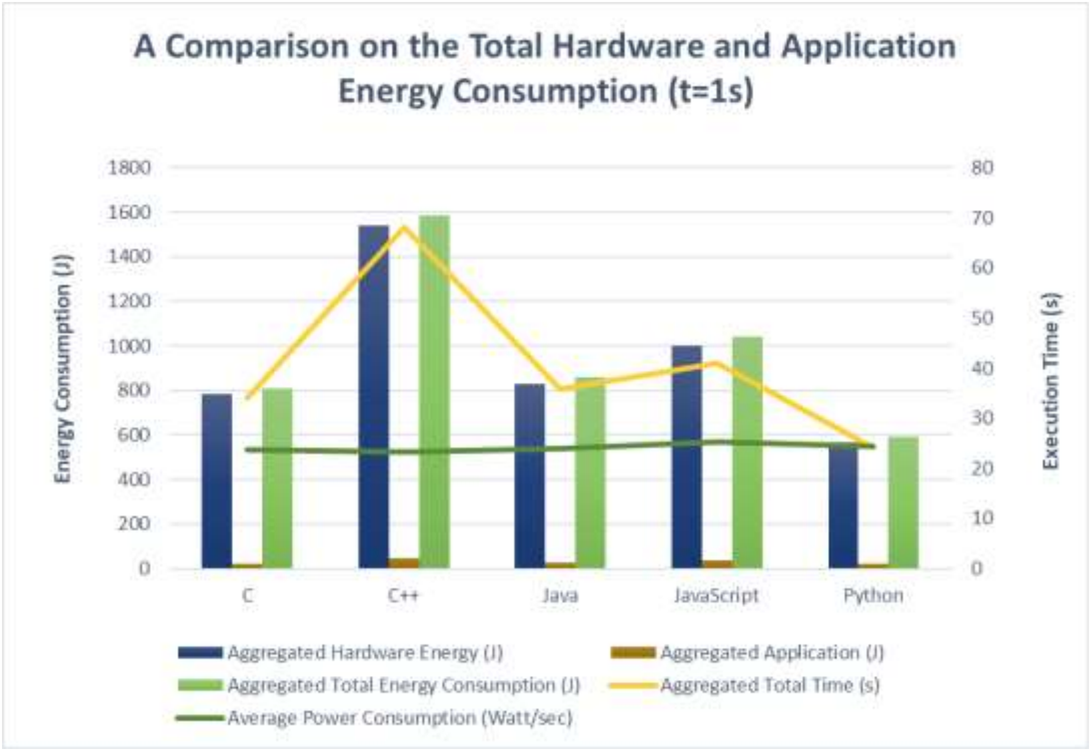


Figure 8. Energy Consumption of C, C++, Java, JavaScript, and Python with Iteration.

The results shown above make it possible to conclude the following:

- The energy consumed by an iterative hello world program varies significantly from one language to another.
- The average power consumption in watt/sec is almost similar between languages. However, the total running time varies significantly.
- The task execution time directly affects the total energy consumed by each language. For instance, C++ has the least power consumption average per second (23.28479 watt/sec). Still, it appeared to be the most energy-consuming language when performing repetitive procedures because it took longer to run the code (68.0425 s). Meanwhile, with the highest power average consumption per second (25.41807 watt/sec), JavaScript came second because it was faster than C++ by 27 seconds.
- The energy consumption of Java and C programming languages in this iteration task shows similar results in the task execution time and average power consumption.
- Python is the greenest programming language in this benchmark. Its average power consumption is relatively high (24.4007 watts/sec); however, it took only 24.2336 seconds to perform one million iterations, the fastest performance.

4.2.2 Results of Recursion Benchmark

The recursive program presents a more intensive CPU task than the iteration. The objective was to test which programming language is energy-efficient in recursion processes. The following table and figure illustrate the energy consumption results across the different hardware and software components.

Table 5. Energy Consumption of C, C++, Java, JavaScript, and Python with Recursion.

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Aggregated Monitor (J)	Aggregated Disk (J)	Aggregated Base (J)	Aggregated Hardware Energy (J)	Aggregated Application (J)	Aggregated Total Energy Consumption (J)	Total Time to execute the task (s)	Average Power Consumption (Watt/sec)
C	85.83012	207.1838	0	517.9594	810.9732	25.2636	836.2368	31.5178	26.5322086
C++	200.2781	614.1195	0.584461	1535.299	2350.281	75.27737	2425.558	83.0645	29.2008993
Java	88.84623	215.6842	0	539.2105	843.7409	28.09276	871.8337	36.4631	23.9100262
JavaScript	117.5554	257.6741	0.284131	644.1853	1019.699	42.35763	1062.057	7.4491	142.575148
Python	99.99767	222.778	0	556.9451	879.7208	49.9568	929.6776	37.716	24.6494212

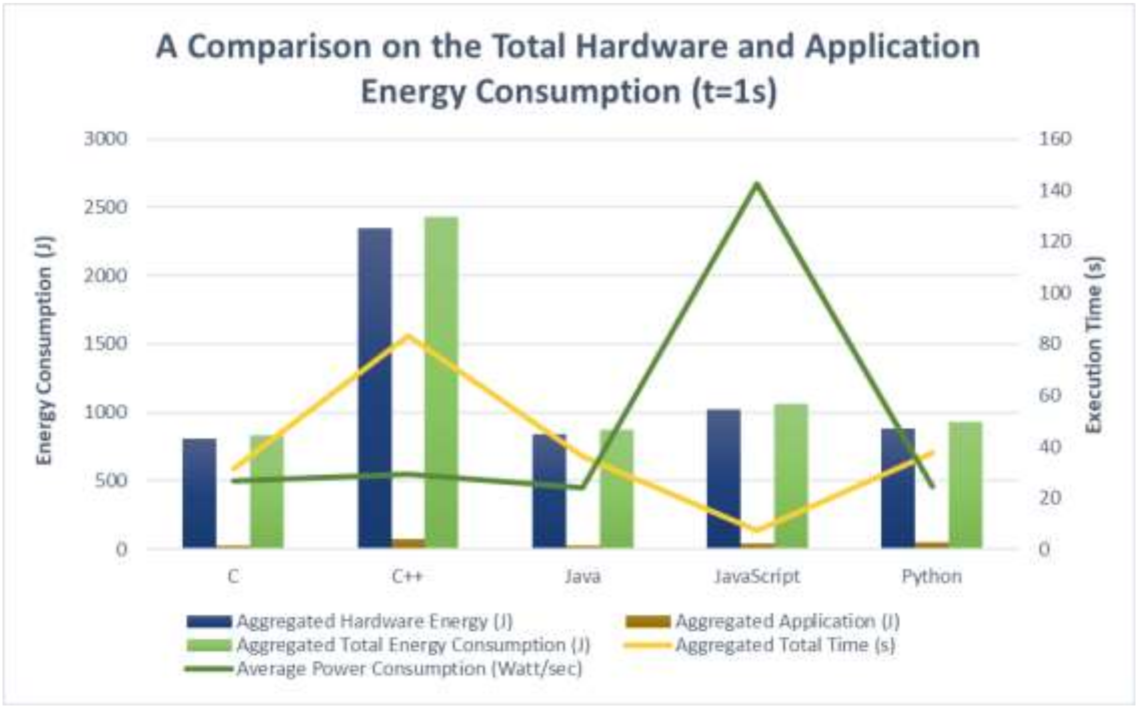


Figure 9. Energy Consumption of C, C++, Java, JavaScript, and Python with Recursion.

The results shown above make it possible to conclude the following:

- The total time required to execute the recursion task and the average power consumption are notably varied across the languages.
- JavaScript average power consumption is almost 400% higher than any of the four remaining languages (142.575148 watt/sec). It consumes a massive amount of power per second to perform recursions.
- Yet, since the total energy consumption is linked to the execution time, JavaScript is not the least energy-efficient because it only takes 7.5 seconds to run the entire code, making it the fastest language. C++ is the most energy-hungry language for recursive operations, with total energy consumption of 2425.558 Joules.
- Python energy behavior experienced some change. Python was the greenest language in the iteration benchmark, but in the recursion benchmark, its energy efficiency degraded. Python recursions execution time (37.716 s) is slower than C, Java, and JavaScript. A remarkable note is that Python aggregated application energy (49.9568 Joules) is the second-highest value after C++. However, JavaScript hardware resource utilization is higher than Python. This led Python to rank third place for this benchmark.
- The C programming language is the most energy-efficient when dealing with recursive algorithms. Furthermore, the energy consumption of Java and C programming languages in this recursive benchmark shows similar results in task execution time and average power consumption.

4.2.3 Results of N-Body Benchmark

The N-body benchmark was presented to further stress the CPU. The objective was to test which programming language is energy-efficient with intensive mathematical operations. The following table and figure illustrate the energy consumption results across the different hardware and software components.

Table 6. Energy Consumption of C, C++, Java, JavaScript, and Python with N-Body problem.

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Aggregated Monitor (J)	Aggregated Disk (J)	Aggregated Base (J)	Aggregated Hardware Energy (J)	Aggregated Application (J)	Aggregated Total Energy Consumption (J)	Total Time to execute the task (s)	Average Power Consumption (Watt/sec)
C	53.36692443	121.3056278	0	303.2641	477.9366	49.57631	527.5129	18.4941	28.52331
C++	51.22563751	144.050182	0	360.1255	555.4013	59.93632	615.3376	19.5525	31.47104
Java	11.87263346	22.32358995	0	55.80897	90.0052	10.42053	100.4257	3.6435	27.56298
JavaScript	13.9353826	24.30040014	0	60.751	98.98678	10.38892	109.3757	4.0486	27.01568
Python	1036.804667	2357.86264	0.189536	5910.288	9305.145	961.4992	10266.64	400.3316	25.64535

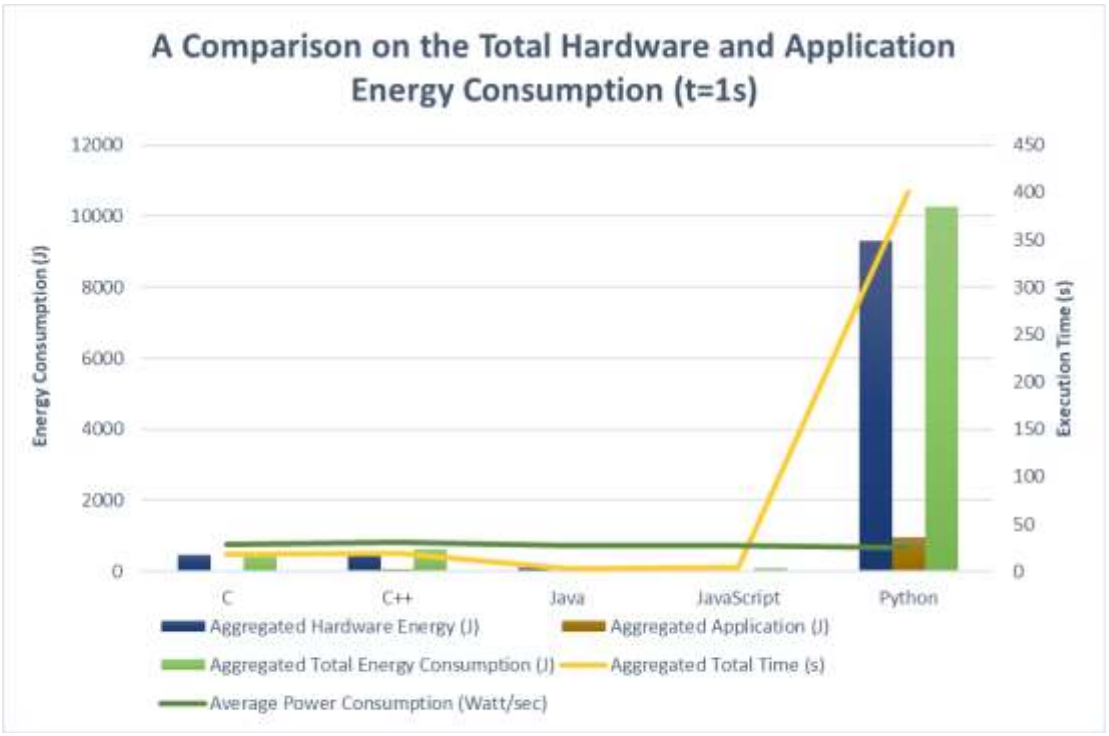


Figure 10. Energy Consumption of C, C++, Java, JavaScript, and Python with N-Body problem.

The results shown above make it possible to conclude the following:

- The most significant result here is Python behavior. It dominates the remaining programming languages as the most time-consuming and energy-inefficient for this benchmark.

- Although the average power consumption per second for Python is the least (25.64535 watt/sec), it took a very long time to execute this intensive mathematical problem; 400 seconds compared to 3.6 seconds for java, the fastest language in this benchmark. This resulted in massive energy consumption; 10,223% higher than Java, the greenest language in this benchmark.

For a better comparison of the energy performance in C, C++, Java, and JavaScript; Python is excluded from the following figure:

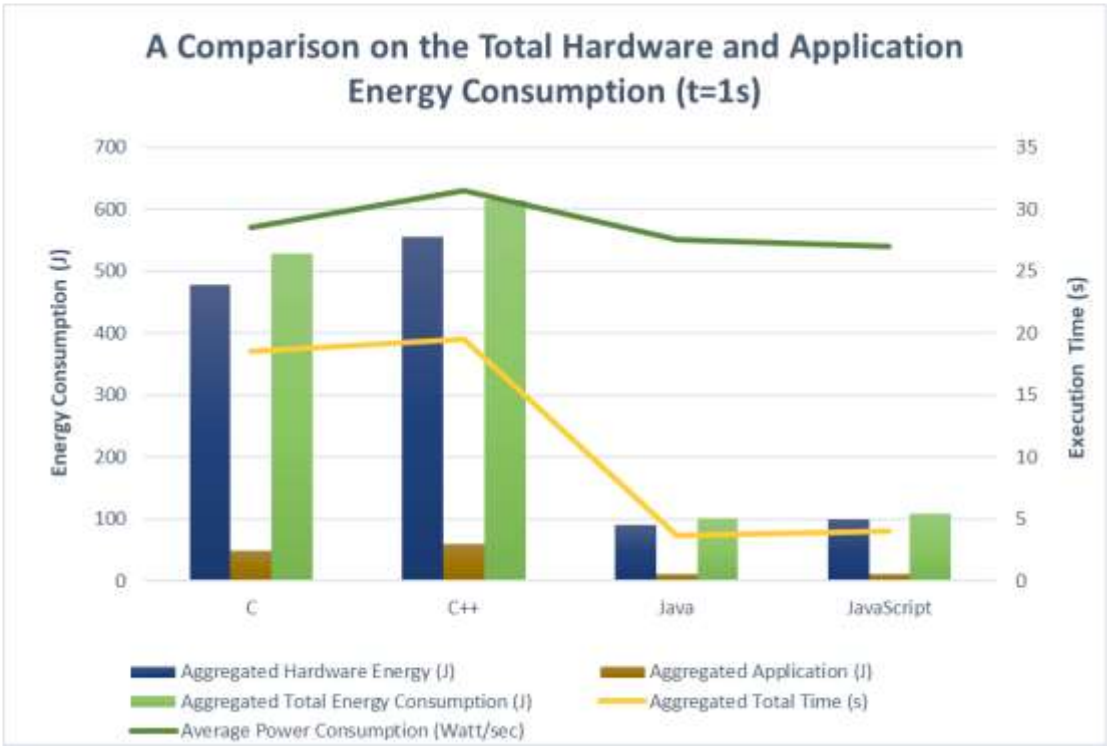


Figure 11. Energy Consumption of C, C++, Java, and JavaScript with N-Body problem.

The results shown above make it possible to conclude the following:

- The total time required to execute the recursion task is remarkably varied across the programing languages and ranged from 19 seconds for C++ to 3.6 seconds for Java.
- The difference in average power consumptions across these four languages is not significant.
- Java is the most eco-friendly language, followed by JavaScript for this benchmark.
- C++ ranked second after python as an energy-inefficient language for this benchmark; there is a huge gap in energy behavior between C++ and Python. However, a significant difference was also noticed between C++ and Java, the greenest, with C++ consuming almost 600% more energy than Java.
- The C language energy behavior experienced some change. C was among the energy-efficient options in iteration and recursion benchmarks, yet, its energy performance degraded compared to Java and JavaScript in this N-body benchmark. It took 18.5 seconds to execute the N-body

problem with an average power consumption of 28.52331 watts/sec to end with a relative consumption level to C++.

4.2.4 Results of Fannkuch-Redux Benchmark

The fannkuch-redux benchmark is our heaviest task, with the highest CPU utilization observed when executing fannkuch-redux codes in the different languages. The objective was to test which programming language is energy-efficient with permutations and accessing indices. The following table and figure illustrate the energy consumption results across the different hardware and software components.

Table 7. Energy Consumption of C, C++, Java, JavaScript, and Python with fannkuch-redux problem

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Aggregated Monitor (J)	Aggregated Disk (J)	Aggregated Base (J)	Aggregated Hardware Energy (J)	Aggregated Application (J)	Aggregated Total Energy Consumption (J)	Aggregated Total Time (s)	Average Power Consumption (Watt/sec)
C	144.7967475	279.6314563	0	699.0786406	1123.506844	118.2746444	1241.781489	42.4572	29.24784
C++	308.6895485	747.6588247	0.110594096	1869.147062	2925.606029	320.8405839	3246.446613	100.2851	32.37217
Java	86.61199339	171.6469879	0	429.1174697	687.376451	71.49584243	758.8722934	28.8534	26.30097
JavaScript	164.0236364	44.14447383	0	110.3611846	318.5292948	133.9417366	452.4710314	7.4491	60.7417
Python	7918.499331	2269.018906	0.265871612	5704.161851	15891.94596	7057.002824	22948.94878	422.361	54.33491

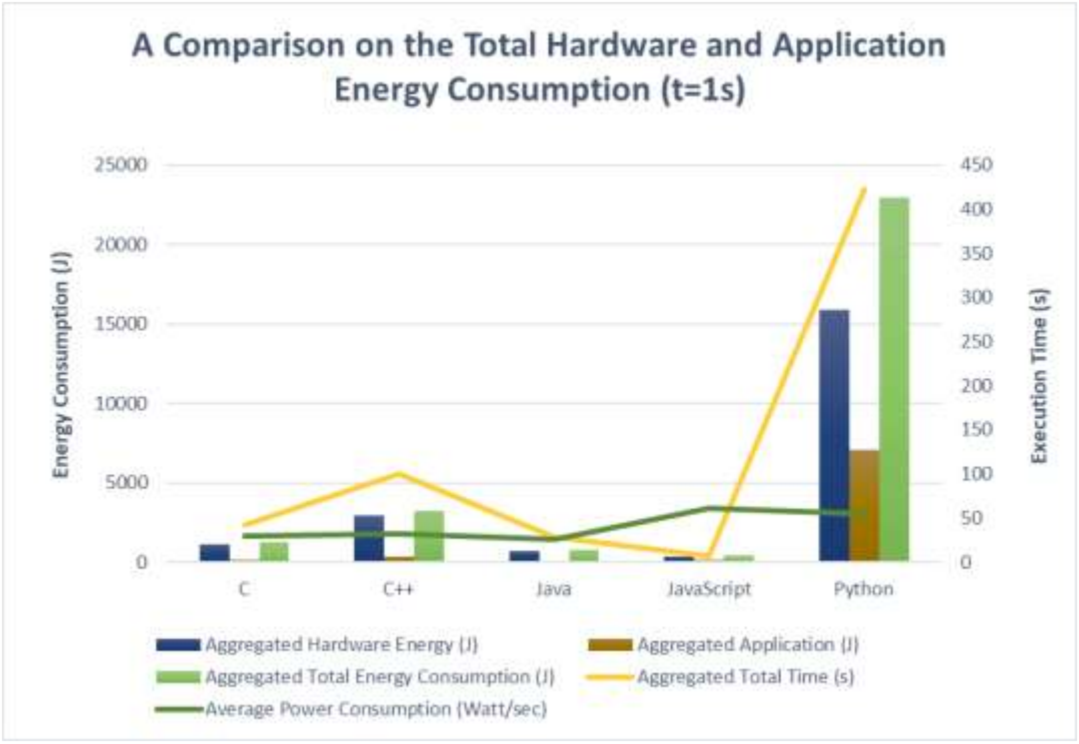


Figure 12. Energy Consumption of C, C++, Java, JavaScript, and Python with fannkuch-redux problem

The results shown previously make it possible to conclude the following:

- Python Energy behavior in this permutation task is the worst.
- The application layer for Python consumes a significant amount of energy compared to all other previous scenarios. The ratio between Application layer consumption (7057.002824 Joules) and hardware layer consumption (15891.94596 Joules) is 1:2. It dominates the remaining four programming languages as the most time-consuming (422.361 seconds) and the most energy inefficient Programming language in the fannkuch-redux benchmark.

For a better comparison of the energy performance in C, C++, Java, and JavaScript; Python is excluded from the following figure:

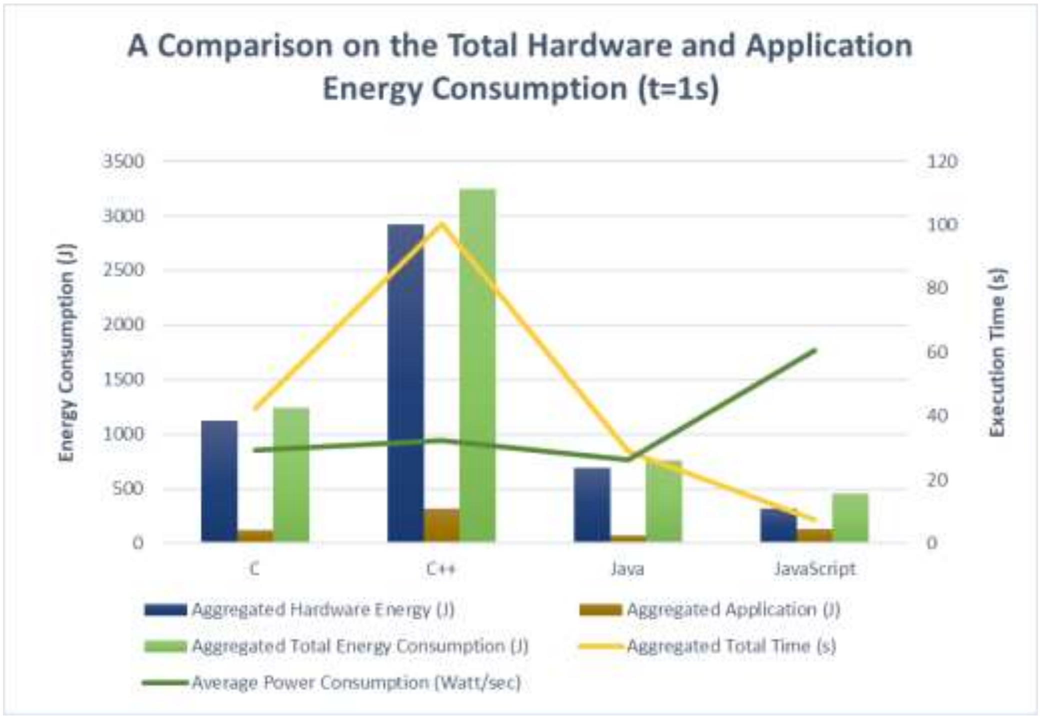


Figure 13. Energy Consumption of C, C++, Java, and JavaScript with fannkuch-redux problem

The results shown above make it possible to conclude the following:

- Both the total time required to execute this fannkuch-redux benchmark and the average power consumed by the four languages varies significantly.
- JavaScript dominates as the most energy-efficient and the fastest language for the fannkuch-redux benchmark, although its average power consumption is the highest across the five languages (60.7417 watt/sec).
- Java is second in the energy efficiency ranking, and C follows Java, making it third.
- C++ ranked second after python as an energy-inefficient language for this benchmark. However, there is a huge gap in energy behavior between C++ and Python. A significant difference is also

noticed between C++ and JavaScript, the greenest, with C++ consuming almost 700% more energy than JavaScript.

The four conducted experiments help in concluding the following:

- Each task running time differs significantly from one language to another.
- As authors (Corral *et al.*, 2014) suggested, the total energy consumption is directly linked to the execution time. However, we can't rely on this only to assume the energy efficiency of a programming language. JavaScript, as a case, was the fastest language to implement the recursion task (7.5 seconds). Nevertheless, it was the second energy-inefficient language with an average power consumption of 142.4 watts/sec.
- Another notable point about JavaScript is that it had the highest average power consumption per second in three of our four benchmarks, although it was very time efficient. This answers our third research question: Is the fastest programming language the greenest?

4.3 Carbon Footprint

This section shows the carbon footprint of conducting our experiments. To calculate how much carbon dioxide was released by each language when executing different algorithms, we used CO_2 conversion factors for the UK electricity published by DEFRA in 2021, available on the following website (*Greenhouse gas reporting: conversion factors 2021*).

Table 8. The carbon footprint of total energy consumption for C, C++, Java, JavaScript, and Python across the various algorithms.

Carbon footprint of Total Energy Consumption (Grams CO2e)	**fannkuch-redux**	**n-body**	**Recursion**	**Iteration**
C	0.073240962	0.031113	0.049322	0.047706
C++	0.191477225	0.036293	0.143061	0.093446
Java	0.044758709	0.005923	0.051421	0.050564
JavaScript	0.026686993	0.006451	0.062641	0.061498
Python	1.35354175	0.605532	0.054833	0.034876

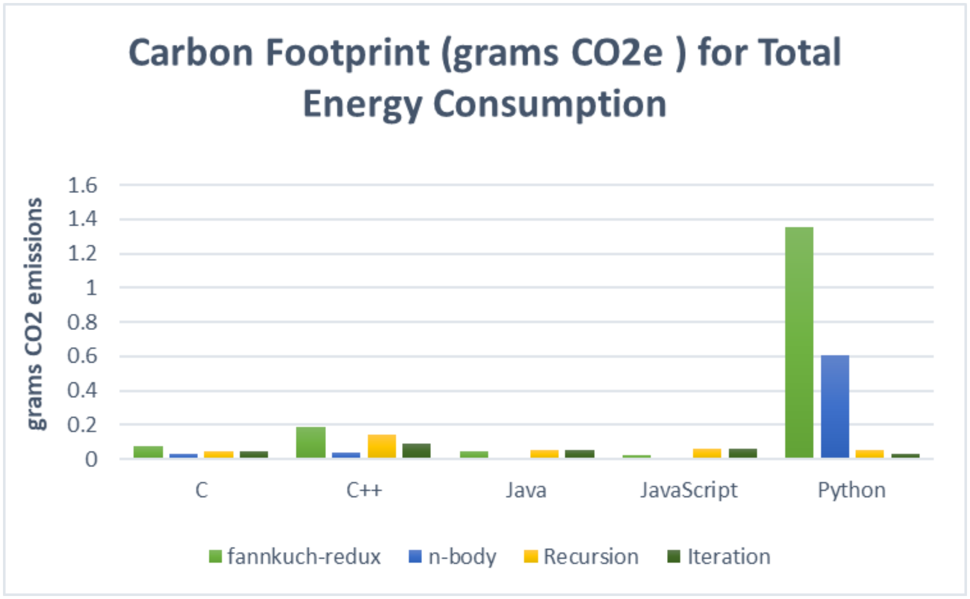


Figure 14. The carbon footprint of total energy consumption for C, C++, Java, JavaScript, and Python across the various algorithms.

The previous charts demonstrate the carbon footprint for each benchmark case. Python CO_2 emissions in two tests (fannkuch-redux and n-body) are exceptionally high compared to other programming languages.

This makes it possible to answer our fourth research question: Is the most popular programming language recently (Python) energy efficient compared to other languages on the list? The answer to this question is: not necessarily true. This variability in the energy performance in Python indicates the language energy inefficiency. With Python's increasing popularity, our results suggest that Python could be a real challenge to all efforts to green the ICT.

Next, to conclude a broad generalization about the energy efficiency across the remaining four languages, six two-sample t-tests were conducted for each pair of these four languages (C, C++, Java, and JavaScript), assuming the following Hypothesis:

- **Null Hypothesis (H_0):** The mean values of energy consumption levels are equal.
- **Alternative Hypothesis (H_a)**: The mean values of energy consumption levels are not equal.

Source Code for T-tests is attached in the Appendix section.

The following Figure shows the Carbon footprint of C, C++, Java, and JavaScript. Followed by a table demonstrating the T-test results.

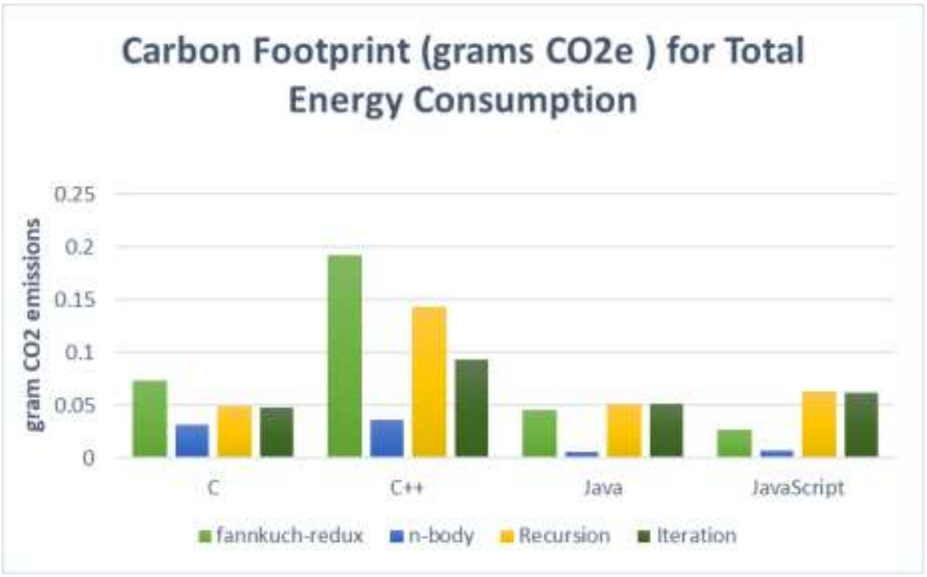


Figure 15. The carbon footprint of total energy consumption for C, C++, Java, and JavaScript across the various algorithms.

Table 9. Two-sample t-test Results

P Value for Two-Sample T-tests							
Programming Languages/Benchmarks		**CPU (W)**	**Monitor (W)**	**Disk (W)**	**Base (W)**	**Total Hardware Power (W)**	**Application (W)**
C and C++	fannkuch-redux Program	5.21918E-21*	2.82092E-31*	0.144936	2.82092E-31*	1.82392E-30*	1.70893E-22*
	N-body Program	0.000133872*	9.94723E-12*	nan	9.94723E-12*	5.39131E-11*	8.39781E-5*
	Hello Program	6.922E-17*	8.29969E-24*	0.330565	8.29969E-24*	1.65048E-23*	2.87018E-16*
	Recursion Program	3.31846E-15*	1.9819E-24*	0.283255	1.9819E-24*	2.77617E-23*	3.67352E-19*
C and Java	fannkuch-redux Program	1.28334E-11*	3.18292E-24*	nan	3.18292E-24*	8.68248E-22*	7.79391E-15*
	N-body Program	8.64885E-16*	1.32407E-21*	nan	1.32407E-21*	2.89422E-21*	8.38382E-16*
	Hello Program	0.0120751*	0.00208383*	0.330565	0.00208383*	0.00190674*	0.000364807*
	Recursion Program	0.293595	0.0206397*	nan	0.0206397*	0.0274127	0.000212147*
C and JavaScript	fannkuch-redux Program	0.0136537*	3.90459E-29*	nan	3.90459E-29*	2.25151E-23*	0.0301381
	N-body Program	4.40401E-12*	8.39757E-23*	nan	8.39757E-23*	2.56239E-21*	1.58551E-16*
	Hello Program	0.00451065*	2.44346E-8*	0.268197	2.44346E-8*	3.40936E-5*	1.62184E-12
	Recursion Program	2.3074E-12*	2.12168E-13*	0.310389	2.12168E-13*	7.5922E-14*	2.19926E-16*
C++ and Java	fannkuch-redux Program	7.13534E-25*	6.72707E-33*	0.144936	6.72707E-33*	7.44011E-33*	4.97739E-25*
	N-body Program	2.4281E-15*	4.4506E-24*	nan	4.4506E-24*	2.81986E-23*	5.06602E-16*
	Hello Program	5.14461E-16*	2.39256E-22*	1	2.39256E-22*	4.34126E-22*	2.06036E-15*
	Recursion Program	8.37703E-15*	6.72829E-24*	0.283255	6.72829E-24*	1.12176E-22*	2.21396E-18*
C++ and JavaScript	fannkuch-redux Program	4.98227E-18*	4.86011E-34*	0.144936	4.86011E-34*	8.85532E-32*	1.32395E-16*
	N-body Program	7.71088E-13	9.33525E-26*	nan	9.33525E-26*	2.13608E-23*	1.75979E-16*
	Hello Program	0.18788	3.97212E-18*	0.268606	3.97212E-18*	1.86107E-11*	7.83027E-05*
	Recursion Program	1.46261E-13*	1.27216E-23*	0.594732	1.27216E-23*	2.88125E-22*	1.17938E-15*
Java and JavaScript	fannkuch-redux Program	1.26671E-10*	2.35672E-24*	nan	2.35672E-24*	5.28653E-18*	5.59928E-09*
	N-body Program	0.447028	0.15095	nan	0.15095	0.185014	0.868615
	Hello Program	0.0103741*	3.69923E-06	0.268606	3.69923E-06*	0.000453637*	5.66781E-11
	Recursion Program	2.05276E-09*	6.4662E-11*	0.310389	6.4662E-11*	7.22359E-11	8.26597E-14*

The Asterisk (*) indicates this P-value rejects the Null Hypothesis for the confidence level of (95%) and alpha value, $\alpha/2 = 0.025$

Carbon footprint and T-tests results emphasize the following conclusions regarding the energy efficiency across the top five most popular programming languages:

- The energy behavior across the programming languages differs significantly with each set of tasks. Python was the greenest in iterations, Java was the most energy-efficient in recursions N-body benchmarks, and JavaScript outperformed others in fannkuch-redux.
- The only exception is for Java and JavaScript Energy performances in the N-body Program. T-tests implied that we could accept our null hypothesis, and hence, for this scenario, there is no significant difference in the energy consumption for all hardware and software components.
- Additionally, Different Energy consumption patterns were observed. Python experienced the highest energy consumption variation across the four benchmarks (from 1.35354175 grams emissions in the fannkuch-redux benchmark to 0.034876 grams emissions in the iteration program). C++ consumption patterns vary across the different tasks also. However, Java, JavaScript, and C showed better consumption steadiness, and energy consumption levels remained within a specific limit no matter what job was being executed.
- The results make it possible to answer our second research question: What is the ranking of these programming languages energy-wise? The answer to that within our scope of work is that it's hard to build a unified ranking for the programming languages based on their energy performance. The programming languages' building blocks are integrated differently, leading to various energy consumption patterns in each task. Nevertheless, it's only possible to make the ranking for each set of computational procedures separately.
- Java, the second most popular programming language, proved to be the most energy-efficient language compared to Python, JavaScript, C, and C++ within our scope of work.
- (Bourdon *et al.*, 2013) and (Couto *et al.*, 2017) suggested that compiled languages outperform interpreted languages in energy efficiency and execution time. Relying on our results, we can argue that this claim is not always valid to the best of our knowledge. Python and Java, interpreted languages, were the fastest in three benchmarks. Moreover, JavaScript showed more energy-efficient performance than C++, a compiled language.
- Finally, these results show the energy consumption at different CPU intensity levels. We can conclude that as the task complexity increases, Python becomes more energy-hungry.

5 Recommendations

The energy performance of five well-known programming languages was investigated by measuring their energy consumption levels in four benchmarks (Iteration, Recursion, Mathematical operations, Permutations, and Indexing). The following recommendations are underlined:

1- The author understands the suitability of some programming languages rather than others in specific domains. However, since the choice of the programming language affects the overall carbon footprint of the software solution, and in cases where Python, Java, C, C++, and JavaScript are comparable, it's recommended to consider Java for its convenient energy performance.

2- As python continues its popularity dominance in the software development field, our results showed that it's pretty energy inefficient for some operations. Our recommendation in this context is for Python designers to put more effort into tackling this challenge for more energy stable releases in the future.

3- Since different programming languages have distinct energy consumption behaviors in various computational tasks, it's hard to build a unified ranking that can suit all the programming scenarios. Hence, it's essential for all authorities concerned about greening the ICT to collaborate to establish a detailed energy-efficiency guideline for the different software development domains.

6 Conclusion and Future Work

Python, Java, C, C++, and JavaScript are the top five most popular programming languages in 2021, according to the IEEESpectrum website. We conducted experiments to rank these programming languages based on their energy efficiency by comparing their energy consumption in four different sets of computational tasks. Using Microsoft Joulemeter and DEFRA carbon footprint conversion factors, our preliminary results show that the choice of the programming language has a significant impact on the overall carbon footprint of the software solution.

In our studied benchmarks, Java and C programming languages consumed less energy than other investigated languages. However, Python, the first programming language on the popularity list, is not the most efficient software programming language, and its energy consumption trends raise concerns when energy-efficient programming matters. These findings can help programmers understand the energy cost of the available programming languages for an energy-aware development tools selection procedure.

This work was limited to four computational tasks, making it hard to form a broad conclusion about the energy efficiency of the investigated languages. Future work is planned on increasing the number of benchmarks by including more computational scenarios to provide a detailed energy efficiency guideline for the programming languages community.

Reference

Ardito, L. and Morisio, M. (2014) ‘Green IT – Available data and guidelines for reducing energy consumption in IT systems’, *Sustainable Computing: Informatics and Systems*, 4(1), pp. 24–32. doi:10.1016/j.suscom.2013.09.001.

Belkhir, L. and Elmeligi, A. (2018) ‘Assessing ICT global emissions footprint: Trends to 2040 & recommendations’, *Journal of Cleaner Production*, 177, pp. 448–463. doi:10.1016/j.jclepro.2017.12.239.

Bourdon, A. *et al.* (2013) ‘PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level’. Available at: https://hal.inria.fr/hal-00850370.

Chistory (no date). Available at: https://www.bell-labs.com/usr/dmr/www/chist.html (Accessed: 17 April 2022).

Corral, L. *et al.* (2014) ‘Can execution time describe accurately the energy consumption of mobile apps? an experiment in Android’, in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*. New York, NY, USA: Association for Computing Machinery (GREENS 2014), pp. 31–37. doi:10.1145/2593743.2593748.

Couto, M. *et al.* (2017) ‘Towards a Green Ranking for Programming Languages’, in *Proceedings of the 21st Brazilian Symposium on Programming Languages*. New York, NY, USA: Association for Computing Machinery (SBLP 2017), pp. 1–8. doi:10.1145/3125374.3125382.

Dmitriev, M. (2002) ‘Language-specific make technology for the Java programming language’, *ACM SIGPLAN Notices*, 37(11), pp. 373–385. doi:10.1145/583854.582453.

Freitag, C. *et al.* (no date) ‘The climate impact of ICT: A review of estimates, trends and regulations’, p. 87.

Gosling, J. *et al.* (2005) ‘The Java Language Specification, Third Edition’, in, p. 688.

Gouy, I. (2008) ‘The Computer Language Benchmarks Game’. Available at: https://benchmarksgame-team.pages.debian.net/benchmarksgame.

Greenhouse gas reporting: conversion factors 2021 (no date) *GOV.UK*. Available at: https://www.gov.uk/government/publications/greenhouse-gas-reporting-conversion-factors-2021 (Accessed: 26 April 2022).

Hamizi, I. *et al.* (2021) ‘A Meta-analytical Comparison of Energy Consumed by Two Different Programming Languages’, in Succi, G., Ciancarini, P., and Kruglov, A. (eds) *Frontiers in Software Engineering*. Cham: Springer International Publishing (Communications in Computer and Information Science), pp. 176–200. doi:10.1007/978-3-030-93135-3_12.

ITU (2018) *ICTs to achieve the United Nations Sustainable Development Goals*, *ITU News*. Available at: https://news.itu.int/icts-united-nations-sustainable-development-goals/ (Accessed: 16 April 2022).

ITU-T, L. 1410 (2014) *Methodology for environmental life cycle assessments of information and communication technology goods, networks and services*. (SERIES L: CONSTRUCTION, INSTALLATION AND PROTECTION OF CABLES AND OTHER ELEMENTS OF OUTSIDE PLANT). Available at: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12207&lang=en.

Kholmatova, Z. (2020) 'Impact of programming languages on energy consumption for mobile devices', in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, pp. 1693–1695. Available at: https://doi.org/10.1145/3368089.3418777 (Accessed: 16 April 2022).

Kienle, H.M. (2010) 'It's About Time to Take JavaScript (More) Seriously', *IEEE Software*, 27(3), pp. 60–62. doi:10.1109/MS.2010.76.

Kumar, M., Li, Y. and Shi, W. (2017) 'Energy consumption in Java: An early experience', in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8. doi:10.1109/IGCC.2017.8323579.

Li, T. (no date) 'Introducing Python'. Available at: https://www.academia.edu/39376646/Introducing_Python (Accessed: 16 April 2022).

Microsoft Joulemeter (no date) *Download.com*. Available at: https://download.cnet.com/Microsoft-Joulemeter/3000-2086_4-75578519.html (Accessed: 25 April 2022).

Mikkonen, T. *et al.* (2007) *Using JavaScript as a Real Programming Language*.

Noureddine, A. *et al.* (2012) 'A preliminary study of the impact of software engineering on GreenIT', in *2012 First International Workshop on Green and Sustainable Software (GREENS)*. *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pp. 21–27. doi:10.1109/GREENS.2012.6224251.

Oliveira, W., Oliveira, R. and Castor, F. (2017) 'A Study on the Energy Consumption of Android App Development Approaches', in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 42–52. doi:10.1109/MSR.2017.66.

Pang, C. *et al.* (2016) 'What Do Programmers Know about Software Energy Consumption?', *IEEE Software*, 33(3), pp. 83–89. doi:10.1109/MS.2015.83.

Pereira, R. *et al.* (2016) 'The Influence of the Java Collection Framework on Overall Energy Consumption', in *2016 IEEE/ACM 5th International Workshop on Green and Sustainable Software (GREENS)*. *2016 IEEE/ACM 5th International Workshop on Green and Sustainable Software (GREENS)*, pp. 15–21. doi:10.1109/GREENS.2016.011.

Pereira, R. *et al.* (2017) 'Energy efficiency across programming languages: how do energy, time, and memory relate?', in *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. New York, NY, USA: Association for Computing Machinery (SLE 2017), pp. 256–267. doi:10.1145/3136014.3136031.

Rashid, M., Ardito, L. and Torchiano, M. (2015) 'Energy Consumption Analysis of Algorithms Implementations', in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–4. doi:10.1109/ESEM.2015.7321198.

van Rossum, G. (1995) 'Python reference manual'. Available at: https://ir.cwi.nl/pub/5008 (Accessed: 16 April 2022).

Severance, C. (2012) 'JavaScript: Designing a Language in 10 Days', *Computer*, 45(2), pp. 7–8. doi:10.1109/MC.2012.57.

Shafique, U. and Qaiser, H. (no date) 'A Comparative Study of Data Mining Process Models (KDD, CRISP-DM and SEMMA)'.

Stroustrup, B. (1986) 'An Overview of C++'. Available at: https://dl.acm.org/doi/pdf/10.1145/323779.323736?casa_token=uuwAP5Abc-UAAAAA:y5BmhaBbI5H_dXzwV0R6At9-B-UFOBQaM8dUcmgjD_Wa8L4YbzK3tqnoUxOPscklLszPLgGHYMU.

Stroustrup, B. (no date) 'An Overview of the C++ Programming Language', p. 23.

Top Programming Languages 2021 (no date) *IEEE Spectrum*. Available at: https://spectrum.ieee.org/top-programming-languages/ (Accessed: 15 April 2022).

Van Hoff, A. (1997) 'The case for Java as a programming language', *IEEE Internet Computing*, 1(1), pp. 51–56. doi:10.1109/4236.585172.

Wallace, C. (1993) 'The Semantics of the C++ Programming Language', in *Specification and Validation Methods*. Oxford University Press, pp. 131–164.

Wikizero - C++ (no date). Available at: https://www.wikizero.com/www/C%2B%2B (Accessed: 17 April 2022).

Wikizero - Java (programming language) (no date). Available at: https://www.wikizero.com/en/Java_(programming_language) (Accessed: 16 April 2022).

Wikizero - JavaScript (no date). Available at: https://wikizero.com/www/JavaScript (Accessed: 16 April 2022).

Wikizero - Python (programming language) (no date). Available at: https://www.wikizero.com/en/Python_(language) (Accessed: 16 April 2022).

Wikizero - The C Programming Language (no date). Available at: https://wikizero.com/www//C_Programming_Language (Accessed: 17 April 2022).

Wirfs-Brock, A. and Eich, B. (2020) 'JavaScript: the first 20 years', *Proceedings of the ACM on Programming Languages*, 4(HOPL), p. 77:1-77:189. doi:10.1145/3386327.

Appendix

Codes for Iteration Benchmark

```c

//Iteration in the C programming language

#include <stdio.h>

int main()
{
  for (int i = 0; i < 1000000; i++)
  {
      printf("Hello world\n");
  }
}
```

```python
# Iterations in Python

for i in range (1000000):
    print ("Hello world")
```

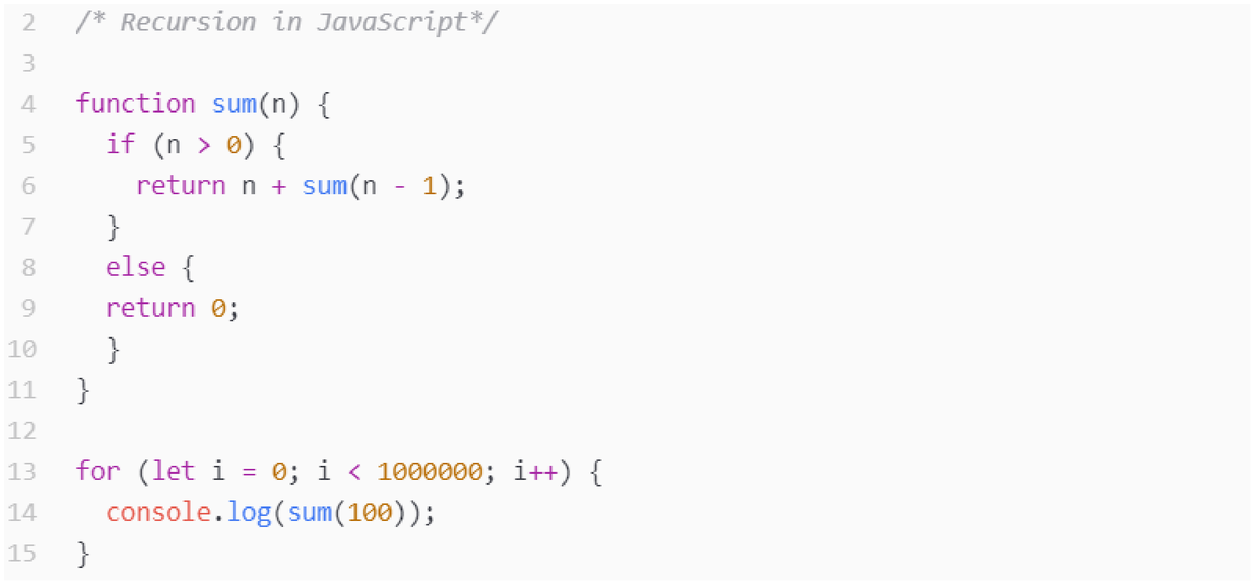
```javascript
/* Iterations in JavaScript */

for (let i = 0; i < 1000000; i++) {
  console.log("Hello world");
}
```

```java
/* Iterations in Java */

public class MyFirstJavaProgram {

   public static void main(String []args) {
     for(int i=0;i<1000000;i++)
     System.out.println("Hello world"); // prints Hello World
   }
}
```

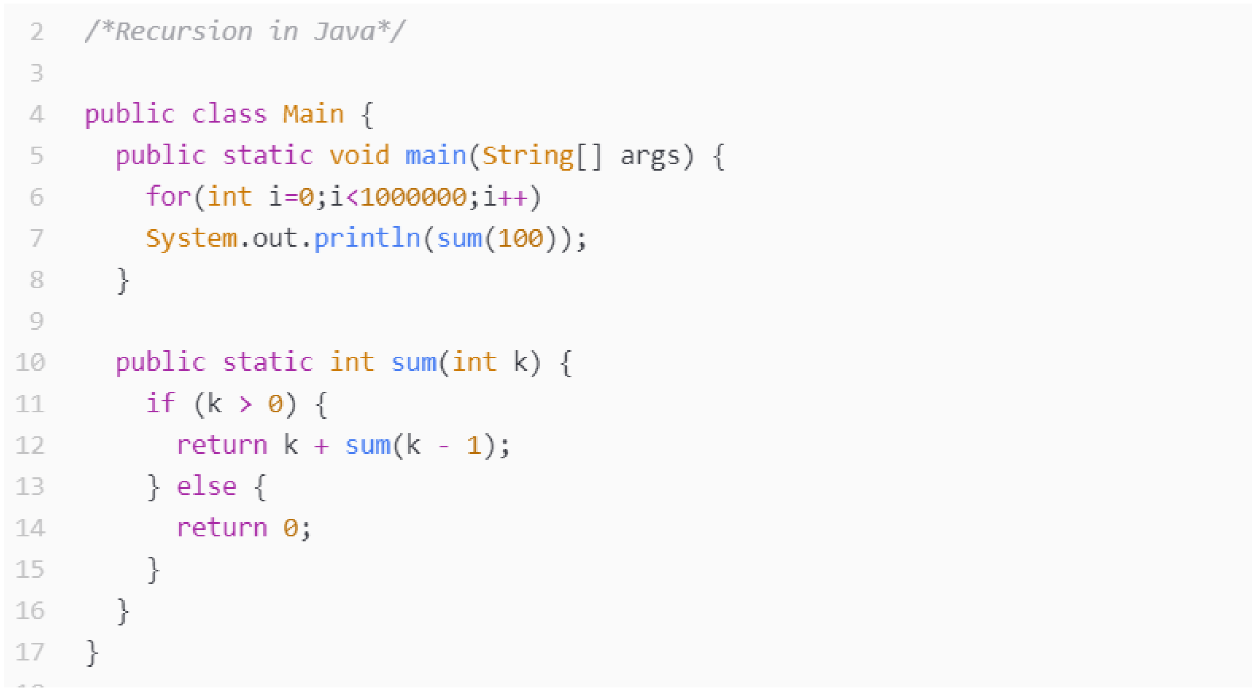
```cpp
// Iteration in C++

#include<iostream>
using namespace std;
int main()
{
    for(int i=0; i<1000000; i++)
    {
        cout<<"Hello World\n";
        }
        return 0;
        }
```

Codes for Recursion Benchmark

```python
# Recursions in Python

def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
  else:
    result = 0
  return result

for i in range(1000000):
    print(tri_recursion(100))
```



```javascript
/* Recursion in JavaScript*/

function sum(n) {
  if (n > 0) {
    return n + sum(n - 1);
  }
  else {
  return 0;
  }
}

for (let i = 0; i < 1000000; i++) {
  console.log(sum(100));
}
```



```java
/*Recursion in Java*/

public class Main {
  public static void main(String[] args) {
    for(int i=0;i<1000000;i++)
    System.out.println(sum(100));
  }

  public static int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
}
```

```c
//Recursion in the C language

#include <stdio.h>

int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}

int main() {

  for (int i = 0; i < 1000000; i++)
  {
      int result = sum(100);
      printf("%d\n",result);
  }
  return 0;
}
```

```cpp
// Recursion in C++

#include<iostream>
using namespace std;

int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}

int main() {
  for(int i=0; i<1000000; i++)
  {
      cout<<sum(100)<<"\n";
      }
  return 0;
}
```

Codes for N-Body Benchmark

The Codes for this benchmark are available in the following links.

- C++ Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gpp-6.html

- C Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gcc-7.html

- Java Source Code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-java-4.html

- JavaScript Source Code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-node-6.html

- Python Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-python3-1.html

Codes for fannkuch-redux Benchmark

The Codes for this benchmark are available in the following links.

- C++ Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/fannkuchredux-gpp-5.html

- C Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/fannkuchredux-gcc-5.html

- Java Source Code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/fannkuchredux-java-1.html

- JavaScript Source Code:

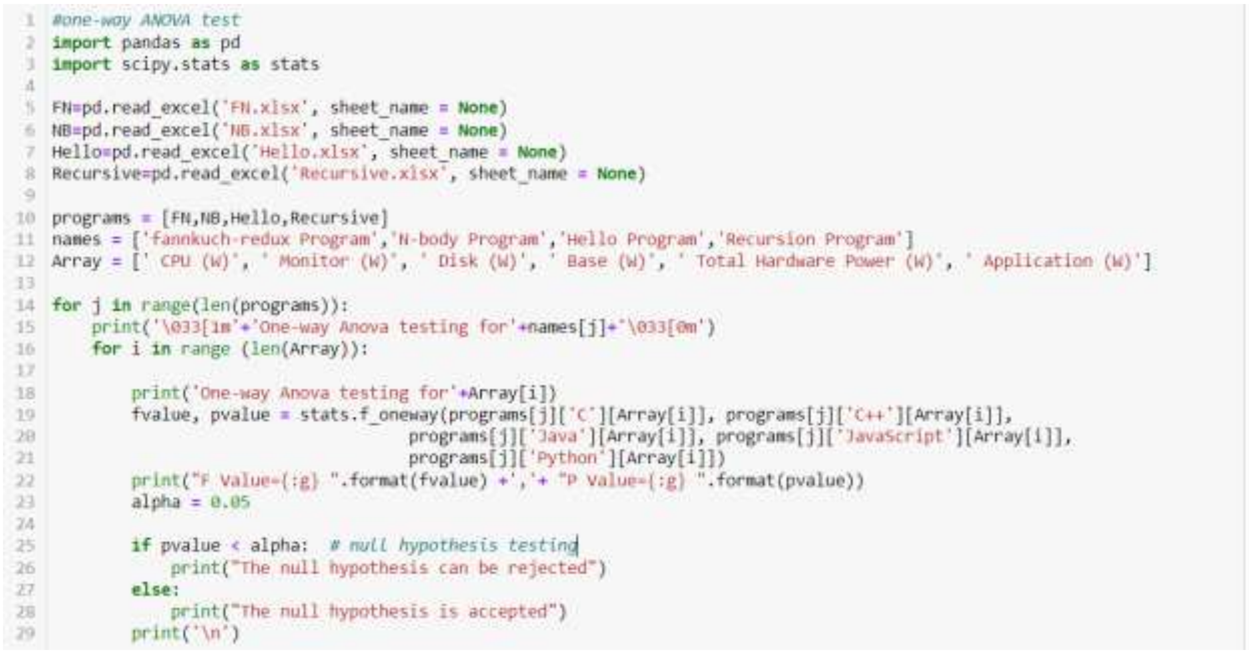
https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/fannkuchredux-node-5.html

- Python Source code:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/fannkuchredux-python3-4.html

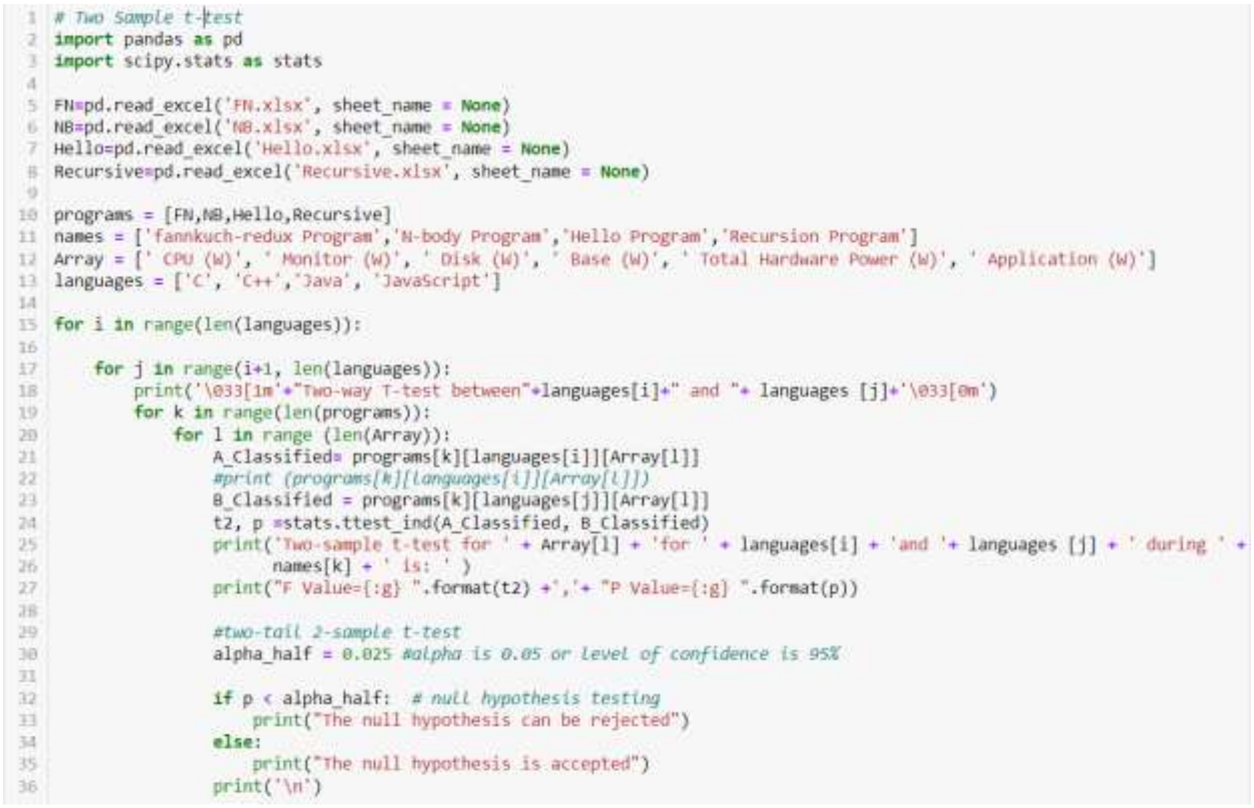
With the support of the Erasmus+ Programme of the European Union

One-way ANOVA test source code



```python
#one-way ANOVA test
import pandas as pd
import scipy.stats as stats

FN=pd.read_excel('FN.xlsx', sheet_name = None)
NB=pd.read_excel('NB.xlsx', sheet_name = None)
Hello=pd.read_excel('Hello.xlsx', sheet_name = None)
Recursive=pd.read_excel('Recursive.xlsx', sheet_name = None)

programs = [FN,NB,Hello,Recursive]
names = ['fannkuch-redux Program','N-body Program','Hello Program','Recursion Program']
Array = [' CPU (W)', ' Monitor (W)', ' Disk (W)', ' Base (W)', ' Total Hardware Power (W)', ' Application (W)']

for j in range(len(programs)):
    print('\033[1m'+'One-way Anova testing for'+names[j]+'\033[0m')
    for i in range (len(Array)):

        print('One-way Anova testing for'+Array[i])
        fvalue, pvalue = stats.f_oneway(programs[j]['C'][Array[i]], programs[j]['C++'][Array[i]],
                                        programs[j]['Java'][Array[i]], programs[j]['JavaScript'][Array[i]],
                                        programs[j]['Python'][Array[i]])
        print("F Value={:g} ".format(fvalue) +','+ "P Value={:g} ".format(pvalue))
        alpha = 0.05

        if pvalue < alpha:  # null hypothesis testing
            print("The null hypothesis can be rejected")
        else:
            print("The null hypothesis is accepted")
        print('\n')
```

Two-Sample T-tests source code



```python
# Two Sample t-test
import pandas as pd
import scipy.stats as stats

FN=pd.read_excel('FN.xlsx', sheet_name = None)
NB=pd.read_excel('NB.xlsx', sheet_name = None)
Hello=pd.read_excel('Hello.xlsx', sheet_name = None)
Recursive=pd.read_excel('Recursive.xlsx', sheet_name = None)

programs = [FN,NB,Hello,Recursive]
names = ['fannkuch-redux Program','N-body Program','Hello Program','Recursion Program']
Array = [' CPU (W)', ' Monitor (W)', ' Disk (W)', ' Base (W)', ' Total Hardware Power (W)', ' Application (W)']
languages = ['C', 'C++','Java', 'JavaScript']

for i in range(len(languages)):

    for j in range(i+1, len(languages)):
        print('\033[1m'+"Two-way T-test between"+languages[i]+" and "+ languages [j]+'\033[0m')
        for k in range(len(programs)):
            for l in range (len(Array)):
                A_Classified= programs[k][languages[i]][Array[l]]
                #print (programs[k][languages[i]][Array[l]])
                B_Classified = programs[k][languages[j]][Array[l]]
                t2, p =stats.ttest_ind(A_Classified, B_Classified)
                print('Two-sample t-test for ' + Array[l] + 'for ' + languages[i] + 'and '+ languages [j] + ' during ' +
                      names[k] + ' is: ' )
                print("F Value={:g} ".format(t2) +','+ "P Value={:g} ".format(p))

                #two-tail 2-sample t-test
                alpha_half = 0.025 #alpha is 0.05 or level of confidence is 95%

                if p < alpha_half:  # null hypothesis testing
                    print("The null hypothesis can be rejected")
                else:
                    print("The null hypothesis is accepted")
                print('\n')
```