To Sustainable Programming

Jaak Henno[1,†] Hannu Jaakkola[2,*,†] and Jukka Mäkelä[3,†]

[1] *Taltech, Ehitajate tee 5 19086 Tallinn, Estonia*
[2] *Tampere University, Pori Campus, P.O. Box 300, FI-28101 Pori, Finland*
[3] *University of Lapland, Rovaniemi, Finland*

Abstract

We have just entered the 'Age of Information', where importance of data/information handling and processing is becoming (far) more important than traditional economy (based on coal and oil) and data/information processing structures have become global. But we are still only beginning to understand the behavior and mechanics of this global 'Information Age' and rules guiding its behaviors, thus also what is needed to make it to produce more Value for the whole Humanity. Development of programming infrastructure is based on the often repeated mantra "Release often, release quick", producing bloatware with enormous number of sub-modules which nobody can understand and control and which just waste electricity executing useless CPU cycles. We have overproduction of IT Tools, but underproduction of useful, working programs and developers/programmers are not able to perform better because of too frequent changes in their tools. Here is considered current situation of popular programming language Python and proposed some measures for improvement.

Keywords

Programming languages, updates, speed of releases, libraries, software documentation

1. Introduction

With exhaustion of Earth's natural resources, we have to behave better. Natural resources are replaced with data – we have to squeeze more from efficiency and innovation. Worldwide spending on Information Technology (IT) is constantly growing, research indicates that investments to IT increase both return on assets (ROA) and return on equity (ROE) [1]. Leading technology research company Gartner predicts that worldwide IT spending grows 8% in 2024 [2].

Especially quickly grows spending on software.

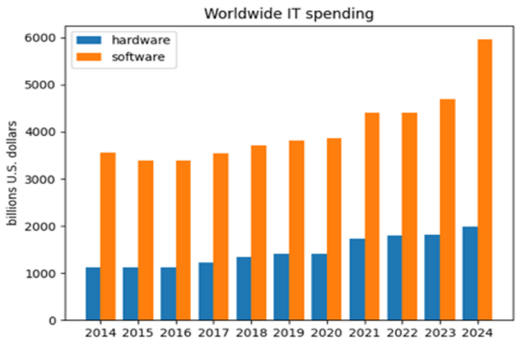

Figure 1. Spending on software is greater and grows quicker than spending on hardware **[3]**.

With growth of the IT has grown also its use of electricity, electricity consumption and carbon impact of AI doubles in every 100 days [4], [5], one ChatGPT query consumes nearly 10 times more electricity as a Google search [6]. Even more troubling is growth in water use for cooling [7]; current

SQAMIA 2024: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 9–11, 2024, Novi Sad, Serbia.

jaak.henno@taltech.ee, hannu.jaakkola@tuni.fi, jumakela10@gmail.com
0009-0008-9886-4734 (J. Henno); 0000-0003-0188-7507 (H. Jaakkola).

trends can make IT from advantageous to detrimental. We should be concerned about sustainable programming.

2. Problems

The Artificial General Intelligence (a concept which still does not have exact formal definition) [8] is like Rossum's Universal Robots [9] before that promising to solve all Humanity's problems, but the rosy perspective of IT 'magical power' is not always true. With modern hardware and money used for IT we should have apps that are faster, smaller, and more robust; every-day life shows that in many cases situation is not improving but vice versa. The iconic game DOOM from 1996 can now run on a pregnancy test [10], but a MATLAB claims that it cannot run on 64bit Windows 7 computer with 8GB if RAM although the Microsoft C compiler what it uses is the same as in Windows 10 computers (and actually Matlab works on Windows 7). The Windows 95 was 50 MB, the Windows 10 – 4 GB – increase in 80 times, but is it 80 times better?

3. Programming and programming languages

Modern world is based on worldwide ecosystem of programs and communication. Programming (mostly) means using some programming language. Currently (mid-2024) the most popular and widely used languages according to the Tiobe [11] and PYPL (PopularitY of Programming Language) [12] indexes are Java and Python languages. Java was in 2005 and 2015 selected as the "Language of the Year"; Python has selected as the "Language of the Year" in 2007, 2010, 2018, 2020, 2021, the lowest position was 13th in 2003.

Figure 2. Popularity of the Python and Java languages according to the Tiobe index [11]

Java and Python are very different languages, but they appeared at the same time (1995-1996), they were used to develop the first version of the Google search engine [13] and comparing their evolution gives insights for the entire programming infrastructure history and development.

Java is a strongly typed object-oriented compiled language. The main idea of Java was to make Java programs universally applicable - w*rite once, run anywhere* - compiled Java bytecode can be executed without recompiling on all hardware that supports Java, i.e. for which has been developed Java virtual machine (JVM). After acquisition of Sun Microsystems in 2009 by Oracle, Oracle is presenting itself as the steward of Java technology, but the Java standard is controlled by the the Java Community Process [14]. Strong typing makes Java very verbose, even the simplest "Hello World!" program uses several LOC (Lines Of Code):

```
public class Exercise1 {
    public static void main(String[] args) {
        // Print a greeting message to the console
        System.out.println("Hello World!");
    }
}
```

Python is an interpreted script language invented at the CWI (Centrum Wiskunde & Informatica) in the Netherlands by Dutch programmer Guido van Rossum. He needed tools for managing the Amoeba operating system, so he designed a simple interpreted script language. The main attraction of the Python language is its simplicity, minimal number of separators, simple visual indication of scoop (by indentation), object-oriented style (no `goto`), variables do not have fixed type and are not declared etc. Python development is coordinated by the Python Steering Council, which currently has 5 members [15]; from 2020 Guido van Rossum is not a member.

In Python the above "Hello World!" program could be expressed with only one line:

```
print ("Hello World!")
```

3.1. Reuse

All programs are reusing pieces of programs generated earlier by somebody. Authors of these pieces are claiming their authority, ownership with copyright, but already from the very beginning of the ''Software age" many people stated, that their programs should be free, available to everyone.

All programming languages use these pieces of free programs as library modules and library packages (groups of modules).

Use of libraries is growing in all languages (nobody wants to 're-invent bicycle'). Libraries call other libraries thus they add enormous number of LOC (Lines Of Code) to the text of the program. For instance, the diagram in Figure 2 is created by 10 LOC of Python 3.11.2 code. This code imports one library `pyplot`, but `pyplot` imports many other libraries. All together are imported 768 libraries which together contain 483056 LOC (and 113 bad libraries which were not found or give error on opening), thus 'the index of visibility' [16] of this 10-line program is 0.0000207.

Programming was once considered as an activity of writing down programming language's commands in proper order. Nowadays programming is mostly importing modules and packages and for many programmers (especially students) loading libraries seem to be the main/only task in creating a new program. Before even analyzing what to do, how to solve a problem are loaded all modules what somehow seem to be related. Libraries contain many files, e.g. the popular Python library `numpy` (numeric Python) contains (under Python 3.11) 1426 files, thus nobody knows exactly, what there is, why it is there, how it works or does it work at all.

Programming languages and their libraries are frequently updated. Updates not only add new features, but also remove or modify existing ones, e.g. in PEP (Python Enhancement Proposal) 594 [17] are presented over 20 modules for removal, in MatPlotlib 3.9.0 release document [18] are more than 30 removals etc.

But some modules may require just these removed ones, thus it may never be possible to have for all modules the latest versions. For instance, Python installer `pip` command to list outdated modules shows three modules:

```
pip list -o
Package     Version Latest Type
----------- ------- ------ -----
ml-dtypes   0.3.2   0.4.0  wheel
protobuf    4.25.3  5.27.1 wheel
tensorboard 2.16.2  2.17.0 wheel
```

Updating `prototobuf` installs its latest version 5.27.1 with warning:

```
pip install --upgrade protobuf
...
ERROR: ... following dependency conflicts.
tensorflow-intel 2.16.1 requires
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3
.20.3, but you have protobuf 5.27.1 which is incompatible.
```

Updating `tensorflow` returns for `ml-dtypes` and `protobuf` the old versions:

```
pip install --upgrade tensorfow
...
```

```
  tensorflow-intel 2.16.1 requires ml-dtypes~=0.3.1, but you have ml-
dtypes 0.4.0 which is incompatible.
  tensorflow-intel 2.16.1 requires tensorboard<2.17,>=2.16, but you have
tensorboard 2.17.0 which is incompatible.
  Successfully installed protobuf-4.25.3 tensorboard-2.17.0
```

Thus there still remain outdated modules:

```
pip list -o
Package  Version Latest Type
-------- ------- ------ -----
protobuf 4.25.3  5.27.1 wheel
```

Use of old modules by some other modules may cause errors and make these modules useless.

3.2. How many Python's do we need?

The programming infrastructure is drowning in updates. The Java and Python languages have together been updated during their existence in 1995-2024 altogether 795 times. For comparison – the ECMAScript (the base for Javascript, i.e. the whole web-page design) has traditionally only one update per year (in June); in the same period June 1997-June 2023 it had only 14 updates [19].

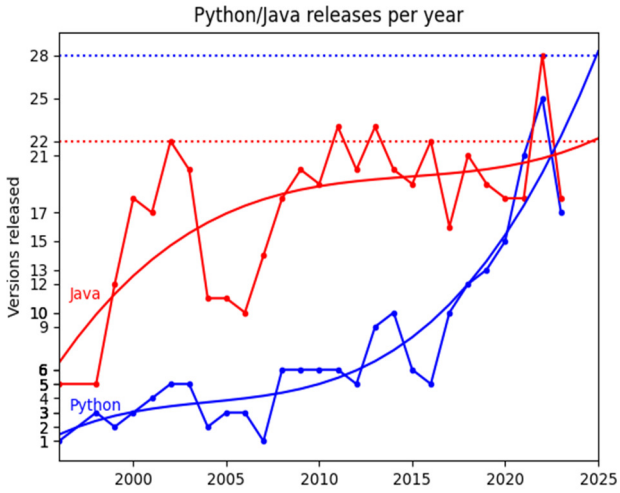

Figure 3. Number of updates per year for the Java and Python languages during 1995-2024 together with third-order approximations for both languages (the smooth curve); data from [20], [21].

The calculated above third-order model predicts, that in 2025 will Python be updated 28 times (an update in less than two weeks) and Java – 22 times (an update in two and half weeks). How could a programming project manager or a teacher in university manage this flow? And the flow is accelerating, these models predict that in 2030 both languages will be updated 49 times, i.e. once in a week.

Java is used on server side, on network and there updates are rarely breaking already existing systems – this would cause lot of problems. Java updates replace the previous version and do not create a new language.

Python users are hobbyists, scientists, students and Python updates are not only for adding new features, the whole language gets a new major version number, becomes essentially a new language. For a major version exist several micro-versions, e.g. currently 5 versions for Python 3.12, 10 versions for Python 3.11 etc.; of those 5 (versions 3.8..3.12) are under development, thus for them will appear new releases (versions). From the Python language created by Guido van Rossum have been derived 26 new languages.

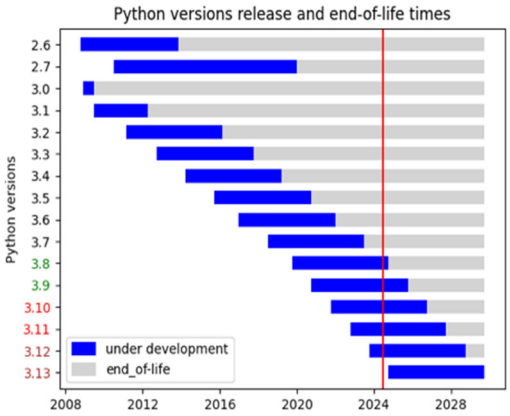

Figure 4. Life cycle of the 26 major versions of the Python languages; by now (June 2024, the red line) altogether have appeared 204 versions [22]; same color (not black) of version numbers indicates the same developer - versions 3.8 and 3.9 are developed by the same developer, versions 3.10 and 3.11 also, versions 3.12, 3.13 also; versions 2.6..3.7 are already stable and release of the next version 3.13 is planned for 1.nov.2024.

Why are at the same time developed five versions of the same product and why three of their five developers are developing at the same time two different versions of the Python language? Different versions were even released at the same day, e.g. versions 1.6, 2.0 and 2.6, 3.0 were released simultaneously.

It is also difficult to understand, how dates and tempo of next releases can be decided for years ahead. The Python "Development Cycle" [23] states:

"new *major versions ... only come when strongly incompatible changes are deemed necessary, and are planned very long in advance*"

How can Python developers know for years ahead, when changes will "*deemed necessary*"? The PEP 361 from 29-Jun-2006 [24] projected 21 releases for years 2008-2013, in PEP 602 from 04-Jun-2019 [25] "*describes a change in the release calendar for Python ...such that feature versions are released predictably every twelve months*". It seems that the holy goal is to fill the whole world with numerous versions of the Python language.

The philosophy of big number of releases was popularized by Eric S. Raymond in his 1997 essay "The Cathedral and the Bazaar" [26]. Raymond stated "*Release early. Release often.*"

Raymond was at the time developer of the open-source mail project `fetchmail`, i.e. a software product that is directly used by customers. For developers of user programs quick releases are logical – users can benefit from new features. But for developers of tools (e.g. programming languages) frequent changes disrupt use of these tool, development of programs for users.

Raymond also added "*And listen to your customers*". Customers voice is definitely the most important feature showing software usability. Python developers consider as index of usability of a package the number of downloads [27]: "*PyPI download statistics are the most powerful tool a Python developer can use to inform their decision making*" [28].

This is a dubious idea: people download just to try out; if they got something useful and working they do not need download again (and also being afraid that next downloads/updates will break already working code). So the big number of downloads of a package may indicate the opposite – users are trying it, but could not get it to work. We tried out the most downloaded packages `boto3`, `botocore` (SDK for Amazon Web Services) in PyPI (Python Package Index) repository. We tried these packages in several computers (several downloads), but could not get it to work. We will try again after some time and thus increase the number of downloads. PyPI considers increase of downloads as indicator of the value of the package. But we could not use packages, thus actually increase of the number of dowloads may indicate that the package is not good.

3.3. Dependencies

New Python modules (often) uninstall some old ones and replace them with older versions, e.g. the updating `tensorflow` to 2.16.1 replaced (besides downgrading the `tensorboard` and `protobuf` modules) also the `numpy` 1.24.4 with older version 1.24.3. But these downgraded modules may also want to downgrade something else, thus soon it becomes impossible to say, what works and what does not work, the only way is to try out.

Most modules and packages do support earlier Python versions, but not farther than four versions back; Python also does not fix bugs further than 3-4 versions back. Thus it has become normal to have in computer installed several versions of Python, e.g. in our test computers are installed respectively Python 3.6.8, 3.7.4, 3.8.0, 3.12.3 (under Win64) and 2.7.18 under Windows 7; there is also a computer with Python 3.11.2 under 64-bit Windows 10. The py-loader can select a version Python, e.g. command "`py -3.11`" loads Python version 3.11.

Every Python language version has its own runtime and library (Python versions can use only their own libraries). Thus the same package gets installed on the same hard disk several times, e.g. the `numpy` package is in our development computer installed 5 times.

Some packages can be used only with some definite version of the Python language. The `tensorflow` package needs for better speed support of gpu - the graphics processor. For use of gpu is needed the CUDA toolkit from NVIDIA [29]. This toolkit depends on the gpu on the graphics card and when the next `tensorflow` version updated their requirements for gpu the `tensorfow` package was not working any more. We can not buy a new graphics card for every update of some package.

The `py` installer can be used in a small script to update all packages/modules under given version of the Python language. Updating all packages updated also `tensorfow`, but the new version requires higher graphics card, thus `tensorfow` become again useless.

To see what is installed in libraries of all installed Python versions we created a small program which traces all `.py` files in a library and counts number of files, which could be read without errors (good modules), total number of LOC in these files and number of files, which create errors on opening (bad modules).

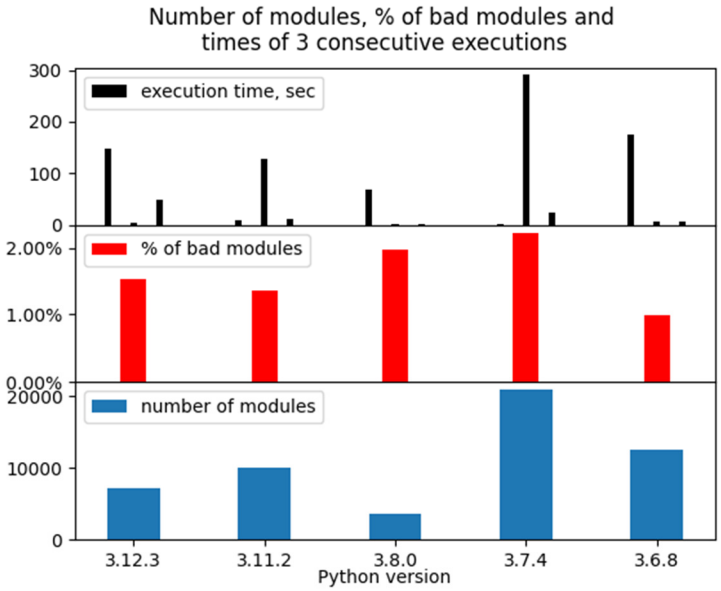

Figure 5. Total number of modules, percentage of bad modules and times of three consecutive executions of the info-collecting program; they differed over 200 times and there was no pattern predicting whether the biggest execution time appears at the first, second or third execution (executing this program was the sole task of the computer).

There have been made many claims about increasing Python speed ("Python 3.11, 10–60% Faster than 3.10" [30], "Python 3.12: Faster, leaner…" [31]). It seems that authors of these claims consider some "ideal" computer, where cpu works without any distractions. But cpu works under Operating System (OS – Windows, MacOS, Linux etc) and can run as quickly as OS allows and use just as much memory as OS gives. Nowadays computers are connected to Internet and OS-s have all the time something to do. At the time of writing this the Tcpview [32] tool shows that several services (and the WinWord program itself) are constantly exchanging information with over 20 hosts. Speaking about increase of speed of Python e.g. under Windows is a very dubious enterprise; besides, Python can use only one core of modern multi-core cpu-s. To investigate we started a program which collects good and bad modules consequtively three times and measured the execution times.

The test indicated that in every Python version there were 1-2% of modules, which give errors on opening (`TypeError`, `UnicodeDecodeError`, `ValueError`, `EOFError` etc); the text representations of Python3.11.2 library modules contained 5551 errors.

Python libraries contain the program `test___all__.py`. Contrary to the name this program responds to errors with `pass` and skips (in Python 3.11) 1447 modules which do not have variable `__all__` - why are these untested modules included in releases ?

Measuring the execution times shows, that times of consecutive executions differ largely in all Python versions. Execution time is essentially reduced by cashe – the function `test___all__.py` was on second execution nearly 5 times quicker.

Bad modules are present in all Python versions and are called also from working programs. We tested the small 10-lines program `it_spending.py` that creates the chart in Figure 6.

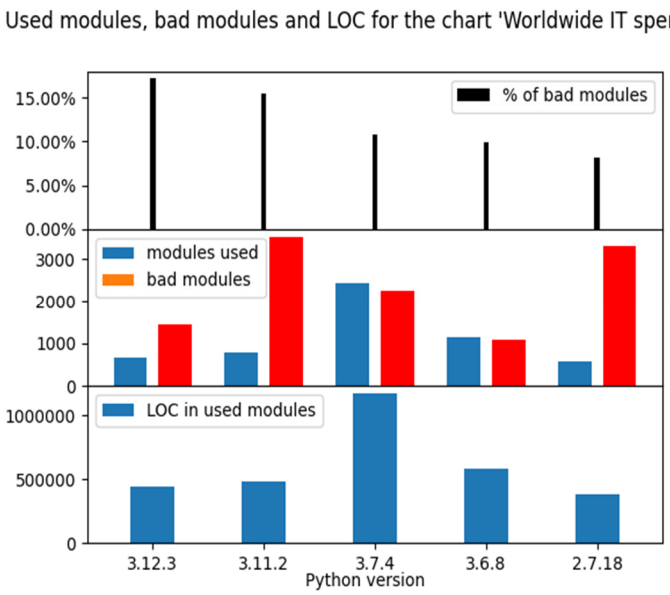

Figure 6. Number of modules used in `it_spending.py` vs number of bad modules.

This test shows that the number of bad modules is greater in newer Python versions (17.7% in Python 3.12.3) and it is much lower in older versions (8.2% in Python 2.7.18).

This again supports the old wisdom "Don't use the latest update" and makes odd Python developers calling language's end of updates "end-of-life" – for all other programming languages these versions are called "stable" and instead "end-of-life" is used "start-of-use".

3.4. Sharing modules and packages ?

The last test indicates that old versions of modules and packages may preferable to the big new ones. The `matplotlib.pyplot` package from all Python versions what we tested draw the chart in Figure 2 without errors. But the `matplotlib` package version 2.5 in Python 2.7.18 contains 557

files, 540 modules (8% bad); `matplotlib` version 3.9 under Python 3.12 contains 779 files, 660 modules (17.1% bad – over twice more), therefore the older versions (especially 2.7) are still widely used.

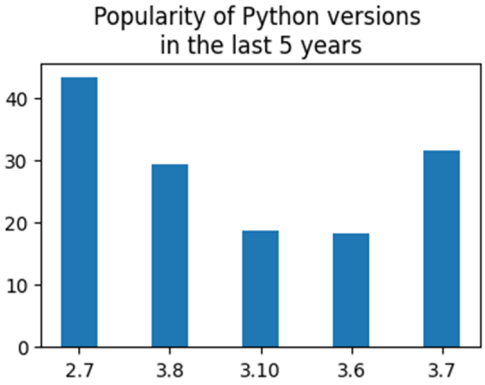

Figure 7. Relative number of Google queries of Python versions during the last five years worldwide; Python 2.7 is still most popular and has less bad modules (see Figure 7).

Thus it seems reasonable to suggest that Python language versions (installed in the same computer) should be able to share packages. Adding to the py-loader ability to load modules from libraries of other versions should not be very complex (updates have modified only Python runtime, the compiler has remained mostly the same) and this would make different versions of Python to look again as one language and not as 26 different languages.

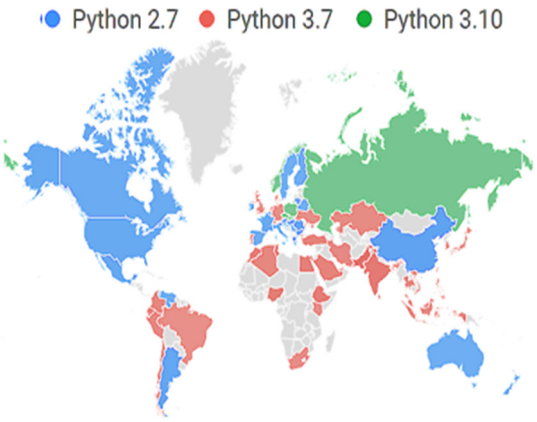

Figure 8. Chart from Google trends shows that in many big countries the most popular version of the Python language is still the oldest – 2.7.

The last two figures show that Python developers announcement "sunsetting Python 2" [33] was premature.

4. Conclusions

The Python language has 26 versions (so far) and lot of libraries which in turn also have many versions, e.g. the boto3 module has 2104 versions/updates [34]. In order to use the language user should install several versions; which all have their own libraries. Python language versions cannot use libraries of another version, although modules from older versions of the language contain fewer bad modules. Python modules cannot be shared by different versions of Python, and Python cannot share different versions of a package using the same runtime. Python user's hard disk is full of dysfunctional and unsupported modules. Does something work or not can be discovered only by

trying; this is a waste of electricity and developer's time and mental energy and the 'broken' state of Python may be the reason why in April 2024 Google fired its entire Python team [35].

Python developers should substantially reduce number of releases of new versions of the Python language and remove from PyPI broken modules. The Python installer pip should not be allowed to uninstall from user's computer some already installed modules and replace them with older versions what is the normal behavior of Microsoft installers.

References

[1] A.H. Hadi, G.R. Abdulhameed, Y.S. Malik, H.H. Flayyih, "The influence of information technology (1T) on firm profitability and stock returns." Eastern-European Journal of Enterprise Technologies 4.13 (2023): 87-93. doi: https://doi.org/10.15587/1729-4061.2023.286212.

[2] 2 Gartner, Gartner Forecasts Worldwide IT Spending to Grow 8% in 2024, 2024. URL: https://www.gartner.com/en/newsroom/press-releases/2024-04-16-gartner-forecast-worldwide-it-spending-to-grow-8-percent-in-2024.

[3] 3 Statista, Information technology (IT) spending Worldwide from 2012 to 2024, by segment, 2024. URL: https://www.statista.com/statistics/268938/global-it-spending-by-segment/.

[4] 4 Goldman Sachs, AI is poised to drive 160% increase in data center power demand, 2024. URL: https://www.goldmansachs.com/intelligence/pages/AI-poised-to-drive-160-increase-in-power-demand.html .

[5] 5 J. Vincent. How much electricity does AI consume? The Verge, 2024. URL: https://www.theverge.com/24066646/ai-electricity-energy-walts-generative-consumption .

[6] Goldman Sachs Research. AI is poised to drive 160% increase in data center power demand. May 14, 2924, https://www.goldmansachs.com/intelligence/pages/AI-poised-to-drive-160-increase-in-power-demand.html

[7] 7 C. Gordon. ChatGPT And Generative AI Innovations Are Creating Sustainability Havoc.URL: https://www.forbes.com/sites/cindygordon/2024/03/12/chatgpt-and-generative-ai-innovations-are-creating-sustainability-havoc/.

[8] 8 OpenAI, Planning for AGI and beyond, 2023. URL: https://openai.com/index/planning-for-agi-and-beyond/ .

[9] 9 K. Capek. R.U.R. (Rossum s Universal Robots), Prague 1920. URL: https://www.gutenberg.org/ebooks/59112.

[10] 10 Popular Mechanics, This Programmer Figured Out How to Play Doom on a Pregnancy Test, 2020. URL: https://www.popularmechanics.com/science/a33957256/this-programrner-figured-out-how-to-play-doom-on-a-pregnancy-test/ .

[11] 11 Tiobe, Tiobe Index for June 2024, 2024. URL:https://www.tiobe.com/tiobe-index/ .

[12] 12 PYPL Github, PYPL Popularity of Programming Language, 2024. URL: https://pypl.github.io/PYPL.html.

[13] 13 G. van Rossum. King's Day Speech, Neopythonic, 2016. URL: http://neopythonic.blogspot.com/2016/04/kings-day-speech.html .

[14] 14 Java Community Process, 2024. URL: https://www.jcp.org/en/home/index.

[15] 15 Hugovk, Python Stteering Council, 2024. URL: https://hugovk.github.io/python-steering-council/ .

[16] 16 J. Henno, H. Jaakkola, J. Mäkelä, Handling Software Icebergs, in: Zoran Budimac (Ed.), Proceedings of the Ninth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, CEUR Workshop Proceedings Vol-3237, 2022, pp. 7:1-7:9. URL: https://ceur-ws.org/Vol-3237/.

[17] 17 Python Enhancement Proposals, PEP 594, 2024. URL: https://peps.python.org/pep-0594/.

[18] 18 Matploplib, What's new in Matplotlib 3.9.0, 2024. URL: https://matplotlib.Org/3.9.0/users/prev_whats_new/whats_new_3.9.0.html .

[19] 19 Webreference, A Brief History of ECMAScript Versions in JavaScript, 2024. URL: https://webreference.com/javascript/basics/versions/ .
[20] 20 Java, JDK releases, 2024. URL: https://www.java.com/releases/.
[21] 21 Python Software Foundation, Python Documentation by Version, 2024. URL: https://python.org/doc/versions/ .
[22] 22 Python Software Foundation, Downloads, 2024. URL: https://www.python.org/downloads/ .
[23] 23 Python Developer's Guide,2024. URL: https://devguide.python.org/developer-workflow/development-cycle/.
[24] 24 Python Enhancement Proposals, PEP 361 – Python 2.6 and 3.0 Release Schedule, 2024. URL: https://peps.python.org/pep-0361/ .
[25] 25 Python Enhancement Proposals, PEP 602 – Annual Release Cycle for Python, 2024. URL: https://peps.python.org/pep-0602/ .
[26] 26 E.S. Raymond. The Cathedral & the Bazaar, O'ReiIly Media, Inc. 2001, ISBN:9780596001087.
[27] 27 PyPI Stats, Most downloaded PyPI packages, 2024. URL: https://pypistats.org/top .
[28] 28 Langui.Sh. Data Driven Decisions Using PyPI Download Statistics, 2016. URL: https://langui.sh/2016/12/09/data-driven-decisions/ .
[29] 29Nvidia Developer, Your GPU Compute Capability, 2024. URL: ttps://developer.nvidia.com/cuda-gpus#compute.
[30] 30 Medium, Python 3.11, 10-60% Faster than 3.10., 2024. URL: https://medium.com/geekculture/python-3-11-10-60-faster-than-3-10-192acb4179f1.
[31] 31 Infoworld, Python 3.12: Faster, leaner, more future-proof, 2024. URL: https://www.infoworld.com/article/3694512/python-312-faster-leaner-more-future-proof.html.
[32] 32 Microsoft, TCPView v4.19, 2024. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/tcpview.
[33] 33 Python Software3 Foundation, Sunsetting Python 2. URL: https://www.python.org/doc/sunset-python-2/.
[34] 34 Pypi.org, The AWS SDK for Python, Releaase history, boto3 1.34.129, 2024. URL: https://pypi.Org/project/boto3/#history.
[35] 35 P. Krill. Google Lays off Python team. InfoWorld Apr 29, 2024, https://www.infoworld.com/article/2336932/google-lays-off-python-team-reports.html.