Make IT sustainable again!

Jaak Henno
Software Technology
Tallinn University of Technology
Tallinn, Estonia
ORCHID: 0009-0008-9886-4734

Hannu Jaakkola
Information Technology and Communication Sciences
Tampere University
Pori, Finland
ORCID: 0000-0003-0188-7507

Jukka Mäkelä
University of Lapland
Rovaniemi, Finland
jukka.makela@urova.fi

***Abstract*—Our 'Information Age' is based on Information Technology (IT), what we expect to create soon more new value that the traditional production. However, in spite of massive investments in IT the expected results are not always seen: opening a web link stalls with "Veryfying link", "We can't check the safety of this website right now", "You do not have right to access folder 'My Documents' " etc. It seems that (one of) the reasons is currently prevailing style of software production – instead of correcting errors in existing software are produced a new version correcting some issues, but incorporating a lot of old 'bad' modules, creating an overflow of versions what is making software increasingly difficult to use. Here has been considered the current state of the Python programming language and proposed some improvements.**

Keywords*—*Python, Java, Library, module, reuse

I. Introduction

In our current 'Age of Information' importance of data/information handling and processing has become far more important than traditional coal-oil based economy. Data and information processing structures have become global and investments on Information Technology (IT) tools – computers and Data Centers are growing worldwide.

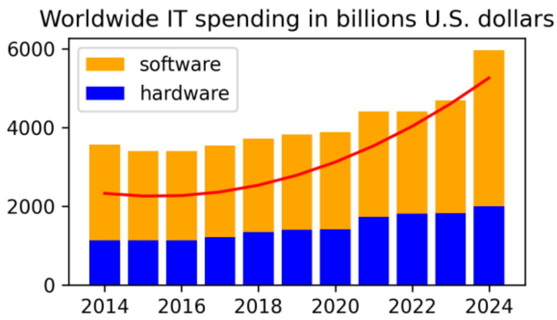

Fig. 1. Non-linear growth of worldwide spending on IT technology [1]

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 and avoid unexpected negative results – overburdening our power grid and water resources.

For a long time computers were considered as very small electricity users, which could always be on. This changed with arrival of chatGPT in November 2022. Its base technology, the Large Language Models (LLM) requires lot of electricity for training. Cost of development of chatGPT was > 4 million USD [2], training the more advanced GPT-4 is estimated to cost 50 times more. A single chatGPT query is ca 10 times more expensive than Google search [3]. The International Energy Agency (IEA) estimates, that the computational power needed for sustaining AI's growth is doubling roughly every 100 days and in 2026 would be nearly equal to electricity consumption of the entire Japan.

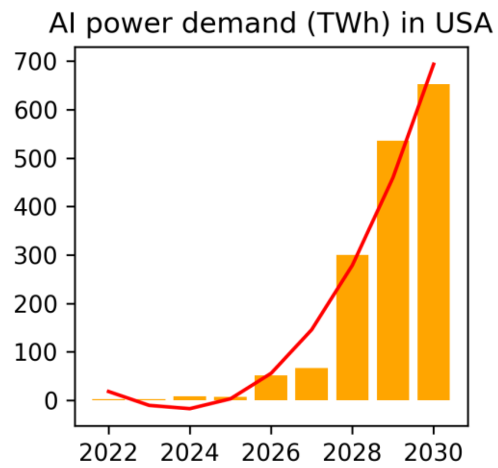

Fig. 2. Growth of power demand in USA; in EU demand will reach 150 TWh in 2026 [4]

Rapidly is also growing use of water for cooling computers in data centers.

Growth in power use is accelerating. In order to satisfy LLM-s demand for processing power are several producers of Central Processing Unit (CPU) increasing power consumption of their new processors, e.g. Intel's new hybrid Falcon Shores is expected to consume 1500W –same as a good dishwasher.

II. Waste of CPU cycles

The money poured in IT has produced some amazing results, e.g. video of the iconic game DOOM from 1996 can now run on a pregnancy test [5], but in many other respects the results do not correspond to our expectations. The Windows 95 was < 50 MB, the Windows 11 – 75 GB, 50 million lines of code (LOC) – increase in 1500 times, but is it 1500 times better?

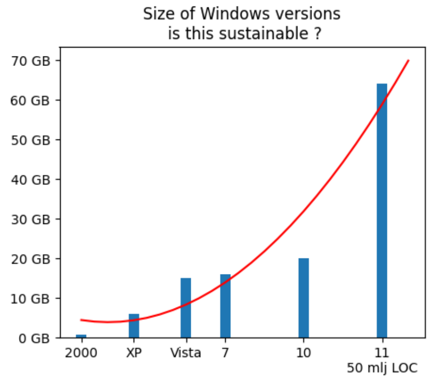

Fig. 3. The size of Windows has grown enormously – is it that much better?

Windows 95 did not announce "You do not have right to access the folder 'My documents'", did not change the decimal separator of my computer and Windows 95 did not install to my computer programs, which are not a part of an operating system such as Copilot or Recall, Windows 95 did not constantly replace quotes to 'smart quotes' even in program text and thus makes program erroneous.

Programming was once an activity of writing Lines of Code (LOC), line by line.

The modern programming is based on reuse - importing libraries, library by library. Programmers often do not know, what exactly is in written by somebody sometimes library, but "*all people use jQuery*", thus we will also use it – although it is useless and only clutters the webpage's code with strange symbols.

Our programs have become monstrous icebergs of hidden in libraries code, e.g. the 10-line Python 3.11 program which draw the chart in Fig. 1 imports only one library `pyplot`, but `pyplot` imports several other libraries, those again others etc. All together are imported 768 libraries which contain 483056 LOC (and 113 bad libraries which were not found or give error on opening), thus 'the index of visibility' [6] of this 10-line program is 0.0000207.

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.

Instead of using bloatware we should be concerned about sustainable programming. In the following is briefly analyzed current state of the popular data-processing language Python.

A. *Python and its libraries*

Currently (mid-2024) are Python and Java the most popular and widely used languages according to the Tiobe [7] and PYPL (PopularitY of Programming Language) [8] indexes. Java was in 2005 and 2015 selected as the "Language of the Year" and has never fallen lower than the 4th place in programming languages popularity index; Python has selected as the "Language of the Year" in 2007, 2010, 2018, 2020, 2021, the lowest position was 13th in 2003.

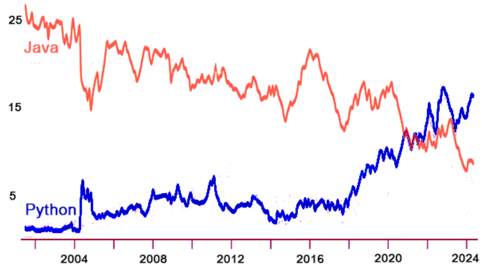

Fig. 4. Popularity of the Python and Java languages according to the Tiobe index.

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 [9] and comparing their evolvement gives insights for the whole programming infrastructure history and development.

Java is a strongly typed object-oriented compiled language, which was developed at Sun Microsystems by a small group of developers lead by James Gosling and released in May 1995 under proprietary licenses, but in 2007 Sun released its JVM as free and open-source software under the terms of the GPL-2.0-only license. The main idea of Java was to make Java programs universally applicable - write 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 [10].

Strong typing makes Java very verbose, even the simplest "Hello World!" program uses several LOC:

```
public class Exercise1 {
    public static void main(String[] args) {
    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 identication), object-oriented style (no 'goto'), variables do not have fixed type and are not declared before use etc. Python development is currently coordinated by 5 members of the Python Steering Council [11]; 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!")
```

B. *Updates*

Besides being the same age and popularity have Python and Java one other similar feature: numerous updates.

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 [12].

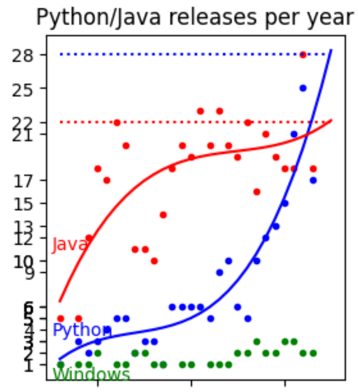

Fig. 5. Number of updates per year for the Java and Python languages and Windows in 1995-2024 together with third-order approximations for both languages (smooth curves); data from [13], [14].

Many computer users curse Microsoft for too frequent updates to the Windows OS ("my computer is a mess after Windows update"), but from the above chart it is clear, that Windows is updated rather sparingly compared to Python or Java. From the third-order approximation of Python releases in the above chart is seen, that in 2025 there will be 28 releases, i.e. one in every two weeks. From 2017 (JDK 9) Java started following the 6 month release cycle, but Java developers still should check for updates (at least) every week [15].

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.

Difference in speed of updates appears in the number of problems what users have – frequent updates create more problems.

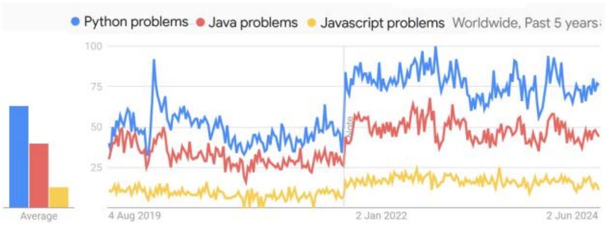

Fig. 6. Google trends show, that Python users have ca 1.5 times more problems than Java users and over 4 times more than Javascript users.

Updates work on Java and Python differently.

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; the languages has (only) 7 versions by now.

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. From the Python language created by Guido van Rossum have appeared 26 new languages.

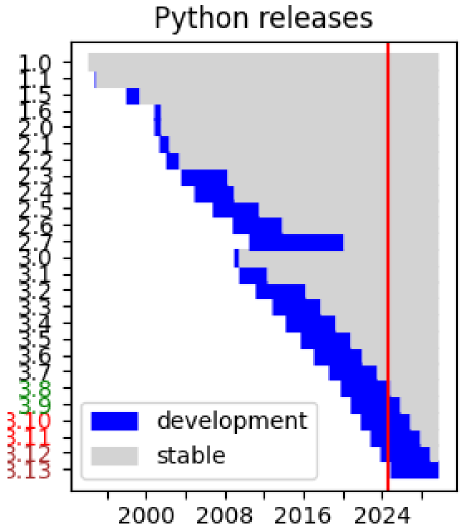

Fig. 7. Python language releases; currently (the red line) are 5 versions under development

The above chart shows only macro versions of the Python language. But for the same macro version exist several micro-versions, e.g. 5 versions for Python 3.12, 10 versions for Python 3.11 etc. Currently three members of the Python Foundation are developing 5 new versions (3.8-3.12), release of the next version 3.13 is planned for 1.nov.2024. Versions 2.6..3.7 are already stable; by Python developers jargon they had reached 'end-of-life'.

By now (July 2024, the red line) have altogether appeared 204 versions of the Python language [16]; twice were simultaneously released two versions (versions 1.6, 2.0 and 2.6, 3.0). In the above chart same color (not black) of version number indicates, that these versions are developed by 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 will also be developed by the same developer.

Frequent updates have become a major feature of modern 'Information Era'. It seems that the mania of updating started from the often repeated mantra "*Release often, release quick*", whitch was popularized by Eric S. Raymond in his 1997 essay "*The Cathedral and the Bazaar*" [17].

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 the use of these tools, 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 use count of a package the number of downloads [18]: "*PyPI download statistics are the most powerful tool a Python developer can use to inform their decision making*" [19].

This is a dubious idea: people download just to try out. If they got something useful and working they do not have to download again (also being afraid that next downloads/updates will break already working code). So the big number of downloads of a package indicates 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 them to work (the reason may be the low number of Amazon cloud servers in Scandinavia and Baltics, see the map [20]). 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 the package, thus actually increase of the number of downloads indicates that the package is not good.

C. *Dependencies*

Updates not only add new features, but also remove or modify existing ones, e.g. in PEP (Python Enhancement Proposal) 594 [21] are presented over 20 modules for removal, in MatPlotlib 3.9.0 release document [22] 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.
Successfully installed protobuf-5.27.1
```

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 will often cause errors and make many modules useless.

Some modules require for use installation of outside (not belonging to the Python language) programs, e.g. (after updating all) Python command

```
import torch
```

creates error:

```
The specified module could not be found. Error loading "C:\Python311\Lib\site-packages\torch\lib\fbgemm.dll" or one of its dependencies.
```

Googling (thus increasing Python popularity index) reveals, that `torch` is searching for file `libomp140.x86_64.dll`, which belongs to Microsoft Visual C runtime. After downloading the file `vc_redist.x64.exe` (14.2 MB) and trying to install Windows 10 announces:

Cannot install a product when a newer version is installed.

The only result: Python developers can consider, that Python popularity has increased by 5 Google searches. And Microsoft tool `Dependency Walker` reveals that besides this `.dll` file there are many other missing dependencies (more than 150) that should be part of the Windows OS.

Some packages can be used only with some definite version of the Python language. The `tensorflow` package needs for better speed support of the graphics processor (gpu). For use of gpu is needed the CUDA toolkit from NVIDIA [23]. 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 do not buy a new graphics card for every update of some package.

Thus it has become normal to install several versions of Python. What works and in which version of the Python language can be established only with testing.

Fig. 8. Python versions installed in the same computer

Versions of the Python language can be handled with py-loader, 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 is installed on the same hard disk several times, e.g. the `numpy` package is in our development computer installed 4 times, `pyplot` – 5 times (it is used also in the Julia language).

All Python versions install some files several times under several names, e.g. in all Python 3.+ are installed `pip.exe`, `pip3.exe` (and in some versions also `pip3.11.exe`), which are the same file with different names.

D. *Bad and missing modules*

The Python version loader `py` can be used in a small script to update all library packages/modules of the Python language. Updating all packages updated also `tensorflow`, but the new version requires higher graphics card, thus `tensorflow` become again useless. Fighting with updates, recalling which packages can be updated and which can't, creating virtual environments – this is clearly waste of human intelligence, a non-sustainable practice.

To see what modules are actually used to create the chart in Fig.1 we created a small program which traces recursively all called `.py` files 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 are missing or create errors on opening (bad modules). This program was used with different versions of the Python language.

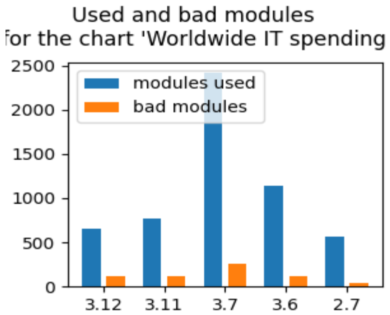

Fig. 9. Good and bad modules in different Python versions

All Python language versions called different number of submodules and in average had 1-2.5% of modules, which were missing or give errors on opening (`TypeError`, `UnicodeDecodeError`, `ValueError`, `EOFError` etc) and the error rate was much higher in some language versions and packages. The text representation (Python function `repr`) of Python 3.11.2 library modules contained altogether 5551 errors. The percentage of bad modules was lowest in the oldest version 2.7 of the language.

But besides of syntactically bad (missing or does not open) there are also many semantically bad modules in Python distributions.

For instance, the well-known Conway's game 'Life' produces error:

```
ModuleNotFoundError: No module named '_curses'
```

Search in Google reveals, that already on Nov 24, 2017 somebody commented "*It's a known bug on windows. It has been open for 5 years*" – i.e. the bug is known already more than 10 years.

Many libraries include a demo program to illustrate its use. The `sdv_demo.py` (in Python 3.11) produces already in the first line error, `Markov.py` enters endless loop, many modules refer to non-existing web pages or documents, e.g. running `test_msccompiler.py` ends with:

```
ModuleNotFoundError: No module named 'pytest'
```

E. *New versions introduce new errors*

The test for counting bad modules called from the Python charting program "Wordwide IT spending" was executed on 2.6.2024; the results are presented in Fig. 9.

When we repeated the test on 4.9.2024 (after several Python updates) the results varied:

Python: 3.6.8 – OK, LOC: 585300, good: 1137, bad: 112;

Python: 3.7.4 – OK, LOC: 1175710, good: 2427, bad: 263;

Python 3.8 - `UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position 308: character maps to <undefined>`;

Python 3.12 - `AttributeError: 'NoneType' object has no attribute 'is_package'`.

The errors were introduced with Python updates.

F. *Sharing modules and packages ?*

The last test indicates that old versions (from previous versions of the Python language) of modules and packages may sometimes be more preferable than the big new ones, if they can perform the same task. 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 – twice more).

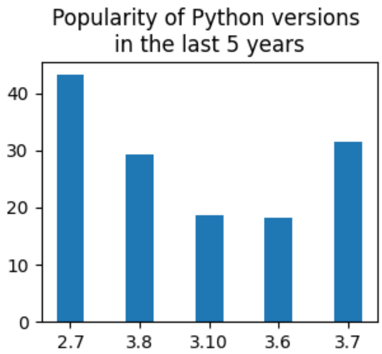

Fig. 10. Google trends csv data indicates, that earlier versions of the Python language (especially 2.7) are still widely used.

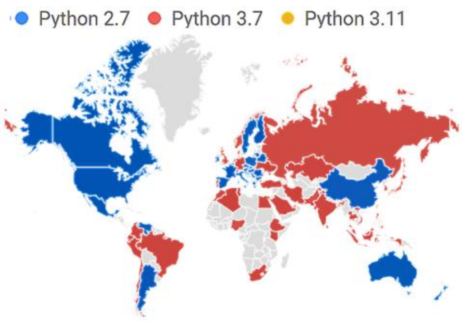

Fig. 11. Chart about geography of users of different versions of Python language shows also, that many big countries still prefer version 2.7.

It seems that the Python developers announcement "SunsettingPython 2.7" [24] was premature.

III. CONCLUSIONS

Thus it seems reasonable to suggest that Python developers should decrease the temp of their attempt to fill the whole world with different versions of the Python language and perform some housecleaning in the PyPI repository of Python packages. It also seems desirable if Python language versions (installed in the same computer) could share packages. Adding e.g. to the `py`-loader ability to load modules from libraries of other versions of the Python language should not be very complex. Numerous updates have modified only Python runtime, the compiler has remained mostly the same (produces similar dictionary with 80 keys in all Python versions) and this would make different versions of Python to look again as one language and not as 26 different languages. Modules and programs with errors just waste programmer's mental powers and electricity executing useless CPU cycles is not a sustainable practice. 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. This causes waste of resources and significantly lowers the value what IT could deliver.

References

[1] Information technology (IT) spending worldwide from 2012 to 2024. https://www.statista.com/statistics/268938/global-it-spending-by-segment/

[2] J. Vanian. ChatGPT and generative AI are booming, but the costs can be extraordinary. March 13, 2023, https://www.cnbc.com/2023/03/13/chatgpt-and-generative-ai-are-booming-but-at-a-very-expensive-price.html

[3] E. Kemene. AI and energy: Will AI help reduce emissions or increase demand? Here's what to know. World Economic Forum Jul 22, 2024, https://www.weforum.org/agenda/2024/07/generative-ai-energy-emissions/

[4] B. Fowke. Before the Senate Committee on Energy & Natural Resources. May 21, 2024.

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

[6] Henno; H. Jaakkola; J. Mäkelä. Handling Software Icebergs. CEUR Workshop Proceedings, https://ceur-ws.org/Vol-3237/, 2022

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

[8] PYPL PopularitY of Programming Languag. https://pypl.github.io/PYPL.html

[9] Guido van Rossum. King's Day Speech. http://neopythonic.blogspot.com/2016/04/kings-day-speech.html

[10] Java Community Process. https://www.jcp.org/en/home/index

[11] https://hugovk.github.io/python-steering-council/

[12] A Brief History of ECMAScript Versions in JavaScript. https://webreference.com/javascript/basics/versions/

[13] JDK releases. https://www.java.com/releases/

[14] Python Software Foundation. Python Documentation by Version. https://www.python.org/doc/versions/

[15] Java updates. https://www.java.com/en/download/help/java_update.html

[16] Python Foundation. Downloads. https://www.python.org/downloads/

[17] E.S. Raymond. The Cathedral & the Bazaar. O'Reilly Media, Inc. 2001,ISBN: 9780596001087

[18] Most downloaded PyPI packages. https://pypistats.org/top

[19] Langui.Sh. Data Driven Decisions Using PyPI Download Statistics. Dec 9, 2016, https://langui.sh/2016/12/09/data-driven-decisions/

[20] AWS Global Infrastructure Map. https://aws.amazon.com/about-aws/global-infrastructure/

[21] PEP 594. https://peps.python.org/pep-0594/

[22] What's new in Matplotlib 3.9.0 https://matplotlib.org/3.9.0/users/prev_whats_new/whats_new_3.9.0.html

[23] Your GPU Compute Capability. https://developer.nvidia.com/cuda-gpus#compute

[24] PSF. Sunsetting Python 2.7. https://www.python.org/doc/sunset-python-2/