Towards a Universal Python: Translating the Natural Modality of Python into Other Human Languages

Joshua Otten
Department of Computer Science
George Mason University
Fairfax, United States
jotten4@gmu.edu

Antonios Anastasopoulos
Department of Computer Science
George Mason University
Fairfax, United States
antonis@gmu.edu

Kevin Moran
Department of Computer Science
University of Central Florida
Orlando, United States
kpmoran@ucf.edu

***Abstract*—The Python programming language plays a large role in computer science today, both in industry and education. While the pseudo-code nature of its keywords and built-in functions/modules makes programming easy to learn for English speakers, non-English speakers do not have this advantage. Our goal is to further the democratization of computer science, allowing anyone to code in their native language, anywhere. This paper describes our vision for realizing this goal by automatically translating Python (keywords, error messages, identifiers) into other human languages, leveraging recent developments in machine translation and language technologies in general. As a first step, we introduce a preliminary multi-lingual Python tool that enables a user to code, translate, and execute Python in 5 additional languages, as well as a roadmap for the future development of our automated framework.**

***Index Terms*—programming languages, Python, democratization, machine translation, programming education**

I. Introduction

Python is arguably one of the most popular programming languages on the planet today [1]. Aside from its strength as an industry and development tool, it is also used extensively in education–specifically for introducing students to computer science and programming. One reason for this is its pseudo-code nature, using English words and phrases that make sense to non-programmers. In principle, rather than memorizing superfluous lists of terminology, acronyms, and abbreviations, a student may focus on understanding the core programming concepts without worrying about language specific details [2].

However, while this works well for English-speakers, students of other backgrounds and nationalities may not receive these benefits. Python's primary strengths of resembling English pseudo-code become nullified when attempted by students whose background is not in English, as they must not only master core programming principles, but learn significant aspects of the English language as well.

This concept is confirmed by several studies which show that students learn programming skills faster when introduced with a coding language based on their native one, and likewise have a more difficult time understanding concepts when the coding is not based on their native language [3], [4]. Furthermore, Github users often write comments and commit messages in non-English languages, demonstrating that there is a demand for multilingual programming [4]. English may

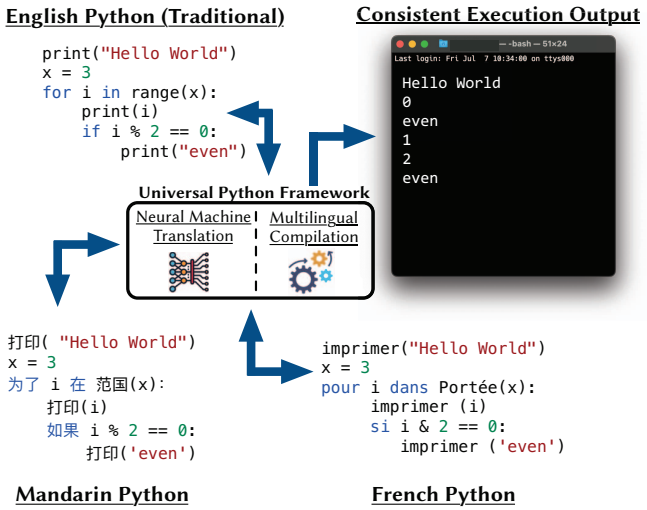

Fig. 1. Concept Illustration for the Universal Python Framework. This framework aims to automatically translate natural language elements of the Python programming language (e.g., keywords, identifiers, error messages, etc.) into various different natural languages other than English.

be *"the lingua franca of the global economy and the de facto standard in cyberspace"* [5] but this *"does not mean that the appeal of operating globally removes the obligation to localize"* [6].

Ideally, one would produce a Python-like language based on each different human language, and maintain and update it as the "core" English Python evolves. Of course, this would be infeasible due to sheer scale and cost if done inefficiently.

Therefore, we hope to meet this need by automatically translating the Python language into other human languages, allowing any number of Python versions to be interpreted and run in the same manner as the standard English Python is used today. This could revolutionize computer science education around the globe, as well as increase programming flexibility in industry and code development.

To put this idea in perspective, if there were an actual universal language translator for the world's languages, then all of humanity's linguistic problems would be solved. There would be no need of a standard, because people could speak

2576-3148/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSME58846.2023.00044

in any language they wish, and everyone would understand them. Similarly, our solution offers such a universal translation system in the context of Python programming. If everyone spoke English, then this would not be necessary. Despite the relatively large number of English speakers in the world (13%) [7], that percentage is still dwarfed by the total of non-English speaking population. Therefore, automated translation offers a much better alternative to computer science education and software development for those who are not fortunate enough to understand English proficiently.

Additionally, we cannot know that English (or at least, modern English) will always be as popular and widespread as it is today. It may be that some other language or dialect will replace English as a lingua franca, and it will be much more beneficial to translate code than for everyone to use a dead or near-dead language just to program computers. Regardless, while our solution has major implications for the evolution of software, its flexibility as a translation system also minimizes any potential downsides of multi-lingual coding.

What we hope to stress is that even though English exists as a programming standard in practice, it is not spoken widely enough to be a truly universal language, and forcing all the people of the world to program with it is not only Anglo-centric, but inconvenient and difficult for those who do not already use it to communicate. The non-English speaking world has to bear an additional burden of learning a foreign language to understand code. In this paper, we discuss the basic principles of our vision with a specific focus on aspects related to implications on software maintenance and evolution.

II. Related Work

A. Language Translators

The idea of translating computer languages is not a new one, and prior work in the compilers domain has successfully accomplished translation between higher level programming languages and lower level machine-interpretable code [8]. What is far more unusual is translating from a computer language defined in terms of one human language, to a computer language defined in terms of a second human language.

Such efforts have been attempted before [4], [9]. For instance, there are a few educational languages that are available in many different human language versions, such as Scratch and Blockly. Additionally, certain widely-used programming languages have versions in one or two other human languages, including a Chinese version of Python [9]. Several other examples existing in the educational space further elucidate the need for non-English languages, such as Glossa[1] and 'KuMir'[2], Greek and Russian Pascal-based programming languages used in secondary education.

Recently, Chris Piech and Sami Abu-El-Haija created a tool called "CodeInternational" that automatically translates parts of Java and Python code into many other human languages. It parses through code, using Google Translate to convert comments, identifiers defined within the codebase, and optionally, string literals, to a secondary language [4]. However, it is important to note that the Python/Java inherent keywords and built-in functions/modules are not translated.

While the advancements made are certainly helpful, to a large degree these have been either incomplete in their translation attempts, or in our opinion not universal enough in scope to truly emancipate programming from its English-dominated standard.

B. Multilingual Language Modeling in Software Engineering

Over the past several years there has been a fast growing interest in and application of various language modeling technologies to software engineering [10]. Initially, these models were largely uni-lingual, trained on single programming languages [11], before models expanded to encompass larger multi-lingual training datasets and tasks [12], [13].

Recent work by Ahmed and Devanbu [14] introduced the notion of *multilingual training* in software engineering. The authors found evidence that training models on multilingual code, that is, code that is written in different programming languages to perform the same function, tends to improve model performance, particularly for languages that are underrepresented (e.g., Ruby). Recent work by Ahmad *et al.* introduced an approach for unsupervised translation between programming languages using back-translation [15].

Additionally, there has been a wealth of work in software engineering research that exploits the bimodality of code (i.e., the fact that software consists of both code and natural language modalities such as comments) to automate various software engineering tasks ranging from comment generation [16] to bug reporting [17].

In the context of software engineering research, the notion of language translation and multilingual language modeling has typically been in the pursuit of the automation of SE tasks and has rarely, if ever, examined the possibility of translating the natural modality of code into various human languages.

III. Universal Python

Our overarching goal is the democratization of programming, by providing a universal Python interpreter that achieves for any language the same benefits Python currently enjoys for English. Students would be able to learn programming and computer science skills using tools modeling their own language. Moreover, if Python can be run efficiently using any language as a base, then industry practitioners could use these tools as well for their own convenience. Programmers from around the world, speaking completely different languages, could collaborate on projects by simply translating code back and forth. In other words, we are looking forward to a new era for Python–no longer dominated by the English-speaking world, anyone can program in any language, anywhere.

To do this, we propose a universal multi-lingual Python interpreter, that allows one to run Python code written in any human language, rather than only English or Chinese.

[1] https://alkisg.mysch.gr/

[2] https://web.archive.org/web/20160112180533/http://lpm.org.ru/kumir2/

IV. Software Development & Maintenance Process

Educational advantages aside, from a software maintenance perspective there are reasonable objections as to whether people should actually be empowered to program in multiple human languages. One concern is that flooding collaborative spaces, such as GitHub, with multi-lingual code would lead to a large number of programs that are unreadable to a majority of developers. One could then argue that English should be a primary standard to avoid such confusion in developer spaces.

However, to a certain extent people are *already* writing code to conform to their native language. For instance, in Java, "among the detected Spanish speakers, 87.2% percent of users write identifiers in Spanish." [4] Statistics like these demonstrate a desire for people to program in a way that is meaningful to them in their own language. The good news is that if our tool works as intended, it would allow seamless translation of code between any two human languages, without much varying efficiency. This means that if UNIPY were to become widespread, it should not matter which language the original version of a piece of code is written in, because it could just as easily be translated back to the preferred language of the developer; this minimizes potential maintenance concerns. When working in a team of people who speak different languages, as long as there is a chosen standard used for the overall project, a member could fetch the given files and convert them all to a different language, then simply translate it back to the original before committing changes.

That said, before UNIPY is widely adopted, we recommend that code written using our tool should be posted in the English version (although not necessarily limited to one language), so that those who do not use UNIPY can easily understand the code. This would also help to avoid potential issues with search engines not recognizing Python code when it is in a non-English language. However, we are careful to not enforce any particular human language version of UNIPY , since doing so would prevent Python from ever becoming truly universal. Later on, in a world where all developers have access to UNIPY , it would not be necessary to post code in English, or any standard, because everyone would have the means to conveniently translate all code to their desired language.

V. Preliminary Work

As proof of concept, we have created a prototype Universal Python, named "UNIPY ", that can be written, translated, and executed in multiple human languages. To do this, we built an interpreter that translates foreign Python code into the original English form, execute it, and return the result. At least at this point, this appears to be a more straightforward approach, than say, attempting to modify the base code and link keywords, etc. to an arbitrary number of possible values. We have also integrated this tool into a web-based application for convenience, which can be found in our online appendix [18].

A. Translating Terms

Since the majority of Python's programming elements consist of English words or abbreviations, we focused our attention on translating these terms into several languages. To do so, we first compiled a list of the 222 standard library built-in functions and modules, and then expanded and unabbreviated each term. With these data at hand, we used a publicly available, state-of-the-art machine translation API to automatically translate the list into several typologically and geographically diverse languages, including Spanish, Sorani Kurdish, and Mandarin. In order to evaluate the robustness of this translation, we had native speakers annotate the results, correcting any mis-translations, as well as abbreviating any items, when needed, to match the generic "pythonic" style.

We first attempted translation with ChatGPT, but while it readily translated Python keywords and function/variable names, it often refused to translate any other built-in functions or modules. In light of this, we opted to use Google Translate for our API. We tested results with five varying contexts embedded within our data. The first used the pure Python terms without any expansion, and the second consisted of these terms, split into separate words (so that `isalpha`, for example, became "is alpha"). For the third test, we translated the unabbreviated phrases that the original Python terms are intended to represent (as in, "is alphanumeric"). The fourth and fifth tests took these expanded terms as part of a larger phrase, to provide the translator more context. The fourth acted as a list of dictionary terms (as in, "is alphanumeric: Returns True if all characters in the string are alphanumeric"), while the fifth used them in an instructive sentence ("In Python, to use the expression that returns True if all characters in the string are alphanumeric, write: is alphanumeric."). Note that, in an effort to present the most accurate context possible, we drew these phrases from W3Schools' descriptions of Python key terms [19]. All in all, our translation results improved significantly after the first two tests, but varied little as a result of the additional built-in context of tests four and five. This demonstrates that abbreviation is a critical factor in automated translation accuracy, and will therefore remain a significant focus of our future work.

We include here a number of results taken from the context used in our third test. Table I displays some examples of terms translated to other languages, and Table II shows the accuracy results of the API for the 222 Python terms of the core library. The quality is quite high for high-resource languages like French and Mandarin with more than 80% of the terms translated correctly out-of-the-box. Of course, lower-resourced languages like Kurdish obtain less than ideal results with accuracy around 30%, but it is important to note that this is only a baseline, with significant room for improvement. We plan to test GPT-like LMS for the translation tasks, but our initial exploration with Google Translate illustrates that we may likely need custom solutions, or at least more elaborate prompting strategies. For example, in the future we plan to use established techniques from domain adaptation [20],

TABLE I
EXAMPLES OF TERM TRANSLATIONS. NOTE THAT THE TRANSLATIONS ARE ABBREVIATED WHEN POSSIBLE.

Python Term	Spanish	Greek	Kurdish	Mandarin
integer	entero	ακέραιη	ژمارەیەکی تەواو	整数
print	imprimir	τύπωσε	چاپکردن	打印
capitalize	capitalizar	κεφαλαιοποίησε	سەرمایەگوزاری	大写
is disjoint	es disjunto	είναι ασύμβατο	ناتەبایە	是不相交的
Syntax Error	Error de sintaxis	Συνταχτικό λάθος	هەڵەی ڕستەسازی	语法错误

TABLE II
CORRECTNESS OF PYTHON TERMS AUTOMATICALLY TRANSLATED TO SEVERAL LANGUAGES.

Language	Accuracy (%)
Spanish	57.7
French	85.1
Greek	60.8
Hindi	51.8
Mandarin	86.0
Kurdish	32.0

[21], to fine-tune a well-performing general-domain translation model using data from the CS/SWE domain so as to better handle programming-related terms. Furthermore, we could fine-tune the machine translation systems on the hand-created translations we produced for the standard library, so that they can perform better when translating terms from other Python packages. This way, we would not need annotators to support additional languages, and we could ideally create support for common imported modules, such as PyTorch and NumPy.

Further in the future, we could incorporate a mechanism for crowd-sourcing using human-in-the-loop approaches, by engaging knowledgeable native speakers to post-edit the automatically translated Python functions and modules on the fly as needed. These corrections could then be used to further improve the model, rendering better-performance for the next iteration of automatic translations, and so on.

B. Interpreting Foreign Code

The lists of human-corrected translated terms allowed us to create and test a prototype interpreter for non-English languages. Currently, our interpreter uses two lists at any given time: a list of the English built-in functions and modules, and a list of the same terms, translated to the second language. Described briefly, our interpreter scans the input code file, searching for matches in the non-English list, and then creates a new `.py` file with the corresponding English functions and modules. Then it executes this new file and returns the output.

In addition, this method supports direct translation for terms of right-to-left languages, such as Arabic and Sorani Kurdish, by inverting the order of terms in these cases. Since all right-to-left code is translated before execution, we do not expect any complications with Python's ASTs. We also note that our translation process is deterministic, so that no code information is lost or changed in translation, and it consistently returns the same result. While we were not able to hand-translate the countless error messages in the Python Language, we can use the same API to automatically translate any error messages that arise. Like the built-in functions and modules, these are stored in look-up tables, so that any issues in translation can eventually be manually corrected. We were careful to avoid translating any quotations or strings, so that the output will remain consistent no matter which language the code is programmed in.

C. Proof-of-Concept Web App

We created a proof-of-concept web app [18], that makes use of a minimal design shown in Figure 2. First, the user can select one of the supported languages. There are two editors, one for English and one for the non-English version. The user may type code into either of the two editor boxes, and translate between them in either direction. Below the editors, two corresponding output boxes display the result of any attempted computation. We also added a "swap" feature that interchanges the code in the editor boxes, along with the selected languages, operating in a manner similar to the functionality of Google Translate. This allows one to conveniently flip between languages, essentially changing the translation direction.

When code is run from either box, our web app links to the prototype compiler, which executes the code in the given language and displays the result in the appropriate output field below. We note that output should remain consistent regardless of selected language, except in the event that execution results in an error message; this is because error messages will be translated and are therefore language-specific.

In sum, code from either box can be executed in various languages to compare the results. We hope this demonstrates that with our new interpreter, someone could code in their native language and run it, just like they might run English Python code.

VI. CHALLENGES

Our primary challenges so far have arisen with regard to Python term translation, and language-specific details. First, English concepts cannot always be conveyed with a single word in other languages. For example, the French phrase for negation (English: "not") is *ne...pas*, where *ne* and *pas* act as delimiters around the phrase that is negated. In Python

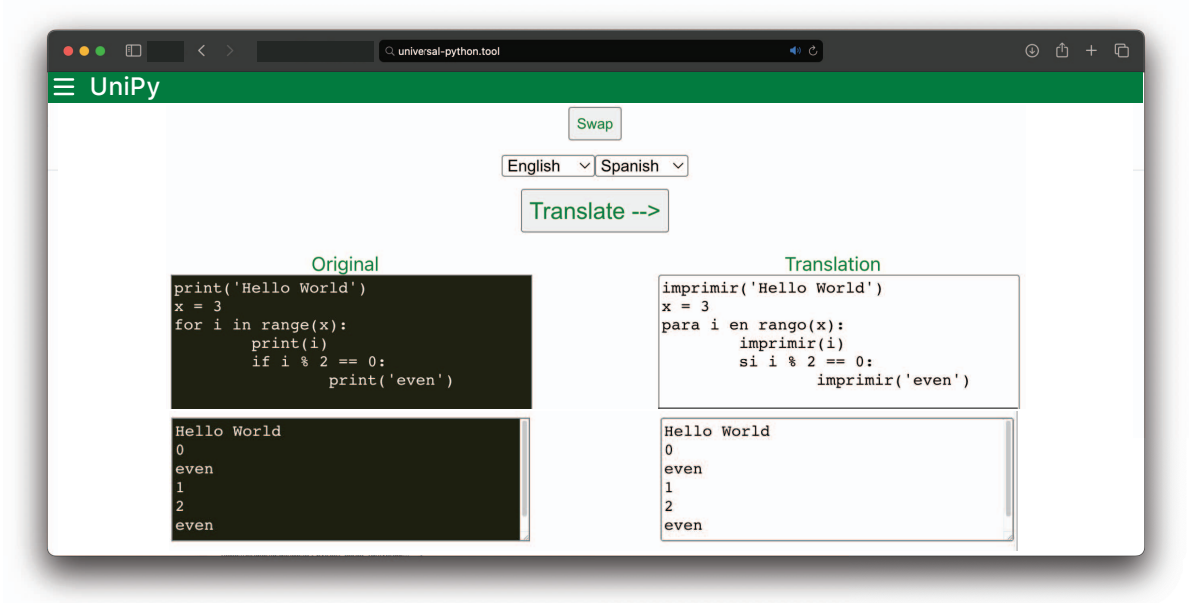

Fig. 2. Example of translating English code to Spanish (editors are the two top panels), along with the output of running both programs (bottom panels). Note that the output is the same.

syntax, one follows the English rule of placing a "not" prior to the to-be-negated expression, but translating to French is not a trivial task, since this requires more than an English-French term replacement. We solved this issue by consulting a native speaker who pointed out that colloquially, the French will often omit *ne* and merely use *pas* prior to the phrase. This allowed us to simply replace the term "not" with "pas" in the French Python syntax. However, similar difficulties may arise with other language versions in the future that will not be as convenient to solve.

Another challenge is handling the abbreviations that already exist as part of library terms. For our preliminary work we experimented with techniques to expand abbreviations (e.g. `chr` to `character`) based on string similarity and co-occurrence statistics in the documentation of the Python core library. But expanding to automatically translating other packages and libraries would require modules that identify abbreviations, as well as multi-word expressions, and appropriately expand them before any translation begins. Put simply, one would first need to realize that `pytorch.linalg` is an abbreviated phrase and expand it to "linear algebra" before feeding it into the translation system. Conversely, a fully-automated solution would perhaps abbreviate, when needed, the longer terms/phrases in the target language. Note, however, that abbreviations are not common in many languages – Indic languages like Hindi and Bengali, for instance, rarely use them. In that case other solutions might be needed (e.g. rephrasing with phrase length in mind) to not make programming in the non-English version too cumbersome.

Other language-specific decisions have to be made before proceeding with translation. This is particularly challenging for languages with rich morphology like Greek, Spanish, or Kurdish. To illustrate this with an example, consider the term `print`. Many English-Python terms do not indicate part of speech or grammatical aspect. As a result, Python function calls such as "print" can be interpreted as nouns or verbs (as in, "the print" or "to print"), but these details are not only absent in Python grammar, they are irrelevant. However, when translating to a language like Spanish, distinguishing between nouns and verbs becomes crucial, as the corresponding words are distinct word forms. Moreover, if "print" is to be a verb, then what should its conjugated form be? One option is the accepted lemma form–the translation one would find in a bilingual dictionary. Should it perhaps be the first person indicative "I print", since the programmer is attempting to print something? Should it rather be "it prints" in the third person, since it is the computer that is doing the printing? On the other hand, it could be a command that the programmer gives to the computer: "print!", in the imperative.

In the end, we opted to utilize the imperative form for examples like this. However, these are all questions that the English version of Python can conveniently ignore. If we are to broaden the reaches of computer science, it is necessary to address these decisions, with the largest challenge being consistency. Any multi-lingual version of Python will work as long as all built-in modules and functions remain internally consistent. However, as we attempt to scale up our Universal Python to a wider range of modules and languages, this will result in growing complexity, and will certainly be a challenge.

VII. Future Work

In the immediate future, we plan to scale our preliminary UNIPY version to include other common libraries, such as PyTorch and NumPy. To do this efficiently with multiple languages will require a degree of specialized automated translation, which we have already started building. The hope is that we can train a model to accurately translate these specific Python terms without the need to hand-annotate each attempt.

This would tremendously improve our ability to expand UNIPY to any language and any module, thus truly entering a world of democratized programming.

With our prototype at hand, the next step will be to evaluate its effectiveness in the educational sector with user studies. The goal will be to measure the degree to which coding in one's native language can improve learning and programming ability. Currently, our general idea includes offering this tool for introductory programming classes in a university setting. Students whose native language is not English could opt to use UNIPY for their assignments, and submit their feedback through a brief survey at the end of the course as to what they thought about the tool, and to what extent having code and error messages in their native language aided them.

Therefore, we propose research questions that concern both technical and human-factored domains. Our human-factor questions include the following:

- To what extent does learning to program in one's native language facilitate the code-learning process?
- Is coding in one's native language more time efficient than coding in a non-native language (assuming one is not already used to coding in English)?
- How much does programming in a foreign language increase overall fluency in that language?
- Is coding in one's native language less stressful than a foreign one, especially while learning to program?

As for technical considerations, we would like to see how multi-lingual code impacts LLMs and other technologies:

- To what extent does multi-lingual program data affect code generation by LLMs?
- Given the link between using code in pre-training and some emergent LLM capabilities (such as multi-hop reasoning), does the language of the natural modality of the code have any downstream effect?

Ultimately, our objective is to allow people from any background and nationality to have the same opportunities for computer science education as English-speakers. Hopefully, with a tool like UNIPY , we can realize this goal.

VIII. CONCLUSION

To summarize, we introduce a prototype framework for Universal Python, that allows for programming and translation between multiple human languages. Our goal is to expand Python's advantages such that non-native English speakers might benefit from them as well. While our preliminary version is functional and an exciting prospect for future programmers, there is still work to be done on its scope and scalability, as well as user studies to determine the extent of its educational advantages. We plan to continue work in these areas, hoping that one day computer science and programming will not be confined to the set of English-speakers, but available to all, regardless of ethnic, cultural, linguistic, or socioeconomic background.

REFERENCES

[1] A. Johnson, "Python popularity: The rise of a language." [Online]. Available: https://flatironschool.com/blog/python-popularity-the-rise-of-a-global-programming-language/

[2] "How is python used in education?" 2021. [Online]. Available: https://www.pythoncentral.io/how-is-python-used-in-education/

[3] B. Hill, "Learning to code in one's own language," 2017. [Online]. Available: https://mako.cc/copyrighteous/scratch-localization-and-learning

[4] C. Piech and S. Abu-El-Haija, "Human languages in source code: Auto-translation for localized instruction," in *Proceedings of the Seventh ACM Conference on Learning@ Scale*, 2020, pp. 167–174.

[5] K. Ohmae, *The next global stage*. Pearson Education Incorporated, 2005.

[6] ——, "Planting for a global harvest." *Harvard Business Review*, vol. 67, no. 4, pp. 136–145, 1989.

[7] Gurmentor, "What is the most spoken language in the world in 2023," 2023. [Online]. Available: https://gurmentor.com/what-is-the-most-spoken-language-in-the-world/

[8] S. Ferguson and R. Hebels, *Computers for Librarians*, 2003. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/language-translator

[9] G. Mcculloch, "Coding is for everyone—as long as you speak english," 2019. [Online]. Available: https://www.wired.com/story/coding-is-for-everyoneas-long-as-you-speak-english/

[10] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. IEEE Press, 2012, p. 837–847.

[11] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 334–345.

[12] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7212–7225.

[13] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8696–8708.

[14] T. Ahmed and P. Devanbu, "Multilingual training for software engineering," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1443–1455.

[15] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Summarize and generate to back-translate: Unsupervised translation of programming languages," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, Dubrovnik, Croatia, May 2023, pp. 1528–1542.

[16] A. Bansal, Z. Eberhart, Z. Karas, Y. Huang, and C. McMillan, "Function call graph context encoding for neural source code summarization," *IEEE Transactions on Software Engineering*, pp. 1–14, 2023.

[17] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM Joint Meeting on the Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 86–96.

[18] J. Otten, A. Anastasopoulo, and K. Moran, "Unipy web tool." [Online]. Available: https://universal-pl.github.io/UniPy/

[19] W3Schools, "Python tutorial." [Online]. Available: https://www.w3schools.com/python/

[20] P. Koehn and J. Schroeder, "Experiments in domain adaptation for statistical machine translation," in *Proceedings of the second workshop on statistical machine translation*, 2007, pp. 224–227.

[21] C. Chu, R. Dabre, and S. Kurohashi, "An empirical comparison of domain adaptation methods for neural machine translation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 385–391.