The International Journal of Engineering and Information Technology

journal homepage:www.ijeit.misuratau.edu.ly

Design and Implementation of Arabic Python Programming Language Interface

Eltaleb, A.
Misurata University, Libya
a.eltaleb@it.misuratau.edu.ly

Ben Sasi, A.
College of Industrial Technology Misurata, Libya
prof_ahmed@cit.edu.ly

Abstract—**This study aims to simplify programming learning for Arabic-speaking students of the Faculty of Information Technology by introducing a simple programming language in Arabic that covers the curriculum of the Programming 1 course. The proposed approach named ArPy (Arabic Python) is rule-based and enables the generation and execution of Python code from Arabic-written code through various steps, including preprocessing, lexical analysis, word mapping, and code generation. Additionally, it provides error detection during code execution. Experimental results have clearly validated that the implemented model (ArPy) is able to efficiently generate functional English Python code from Arabic code with a similarity accuracy of about 87%. Furthermore, ArPy has received positive feedbacks from the target users bridging the language barrier in the field of programming education in Arabic countries.**

Index Terms—**Arabic programming, language interface, natural language processing, Python, programming education.**

I. INTRODUCTION

Programming's evolution closely aligns with computing progress and development. Early computers operated through complex binary machine language, posing challenges for human understanding. Then, higher-level programming languages emerged, making code more human-friendly. Thus, compilers are able to translate code and detect syntax errors in these languages [1, 2]. Achieving programming expertise demands experience, tool mastery, problem-solving skills, and strategic design [3]. However, English predominance hinders non-native speakers in programming education. Programming languages are predominantly written in English, which poses a significant barrier to entry for non-English speakers [4]. Programming languages, documentation, and online resources are primarily available in English, making it difficult for non-native English speakers to understand programming concepts, navigate the subject's complexities, and contribute to the programming community [5, 6]. Consequently, Arabic speakers in Libya for example face challenges in their programming education [7].

Received 02 May , 2024; revised 11 May, 2024; accepted 15 Mar 2024. Available online 08 Aug, 2024.

Swidan and Hermans' study [8] explored the design of programming languages for non-English speakers, focusing on key aspects like keyword translations, variable naming, numeral handling, punctuation integration, right-to-left language support, and multilingual programming facilitation. However, limited research exists on programming in Arabic using Python, a beginner-friendly language [9]. Moreover, existing methods lack flexibility in handling different writing styles and may not effectively translate Arabic words into code, particularly in the context of Python programming [9]. To our knowledge, there is no comprehensive, no easy to use, flexible, and accurate Arabic programming language has been developed.

Thus, it is necessary to create a simple programming language in Arabic for beginners of IT students covering the curriculum of Programming 1 course. This can be done by providing keyword and error messages translations into Arabic language [8]. By making programming more accessible to Arabic speakers, we can promote inclusivity and diversity in the field of programming.

This research aims to address the challenges of programming language education for Arabic speakers by developing an Arabic Python programming language interface named ArPy (Arabic Python). The interface will be designed to be flexible and handle different writing styles, effectively translate Arabic words into Python code, and support debugging by simplifying and translating different debugging errors into Arabic. The proposed programming language interface will make programming more accessible to Arabic speakers, especially beginners. It will also increase the number of Arabic speaking programmers, which will benefit the Arabic-speaking community and the global technology industry.

The remainder of this paper is organized as follows: Section II discusses some related work of Arabic programming languages. Section III presents our model named ArPy; whereas section IV conducts an extensive experimentation of ArPy using several criteria. The final section concludes this paper and discusses future research directions.

II. RELATED WORK

Due to the importance of this subject, as English language constitutes an obstacle in teaching programming for those whose original language is not English, many researchers around the world have developed programming languages in the native languages of their countries, such as: Arabic, Spanish, and Russian, etc. In the literature review, we explore studies on programming languages designed for non-Arabic speakers. We have categorized these studies into two categories: non-Arabic and Arabic programming languages.

Beseiso et al. [10] (2010) conducted reports on the survey of the support for Arabic in some of the existing Beseiso Semantic Web technologies, and gives future scenario in applying Semantic Web for Arabic applications. Finally, multilingual support for these new technologies is also discussed.

Elazhary [11] (2012) developed Arabic versions of LISP and SQL in an attempt to figure out whether developing versions of common programming languages, that are like natural languages of programmers would improve their programming capability. This research developed translators that can translate programs between the corresponding Arabic and English versions of these programming languages for portability. The paper explained the Arabic version of SQL.

In the work of Bassil and Barbar [12] (2012), MyProLang was introduced as a programming language that combines GUI and Arabic natural language, with similarities to C++. MyProLang simplifies coding through features such as include-directives, compiler primitives, class declarations, and functions. It employed GUI template-driven source-code generators, displaying potential for improving productivity, software quality, and time-to-market in software development. However, it's worth noting that the GUI component might not be the most effective tool for students aiming to grasp fundamental programming concepts.

Othman [13] (2016) have built DHAD, an Arabic programming language and compiler aimed at helping students learn C and Assembly programming. DHAD enables coding using Arabic keywords and translate them into C and Assembly languages. It features its integrated development environment and a "Help" option in Arabic. The language also includes an Arabic Programming Language (APL) extension with high-level language-like keywords. The programming language offers numerous keywords, empowering users to customize their format and change any keyword within the DHAD program. However, C and Assembly are compiled languages, which might pose development challenges even when using Arabic.

Bassil [14] (2019) published research which he discussed designing a new programming language called Phoenix. It is a computer programming language that is high-level, object-oriented, compiled, and Arabic. The syntax of its wording is exactly similar to the C# language and is supported by an Integrated Development Environment. A prototype software written using Phoenix and its equivalent implementation written using C# were seen in the experiments. Results have shown Phoenix's various strong characteristics, including functions, while-loop, and arithmetic operations.

Almanie et al. [15] (2019) have developed an edutainment application to educate the fundamentals of Python for Arabic speaking children. The authors aimed to enhance the digital fluency of Arab children, empowering them to create, design, and innovate with new media, moving beyond passive Internet activities like texting, playing, browsing, and online interaction. Python was chosen since it is a user-friendly and highly readable programming language known for its concise code compared to languages like C++ or Java. However, this application is exclusively designed for kids.

Python is a beginner-friendly and widely-used language known for its simplicity and readability [16, 17]. Thus, Python language should be more suitable for education purpose especially for beginner programmers. Python is an interpreted language which simplifies the development and debugging phase for programmers compared to compiled and low-level programming languages such as C++, C, and Java [17]. However, the interpreted language may be less efficient in big projects compared to the compiled ones, but it remains suitable for education purposes. Existing methods primarily focus on Java, C++ and C languages [18].

III. METHODOLOGY

The front-end of our Arabic programming method (ArPy) comprises preprocessing, lexical analysis, and dictionary word mapping. It involves mapping Arabic words into Python keywords to handle input code and adapt to different writing styles. The back-end focuses on code generation, producing Python code that can be executed by machines.

In this research, we have focused on Arabic language to simplify the learning of the basics of the programming for beginner Arabic students who find it difficult in the early stages. To overcome the problem faced by these students, this research suggested a programming language that uses the Arabic language to write commands instead of the usual English language, according to the rules of the programming language. The language model was designed and implemented using Python programming language as a fundamental tool to develop the concept and for prototyping. Also, we used this language to implement a graphical interface where the user is able to test the model. Thus, when the user writes an Arabic code according to some simple rules, the program will be able to translate it and run it as a Python code using a number of steps. The procedure of the implemented ArPy model is shown by the block diagram presented in Figure 1. The input code in Arabic passes by four stages which are: preprocessing, lexical analysis, Arabic word mapping, and code generation. Finally, the output is executed as a Python program where the errors can be identified and translated in Arabic language.

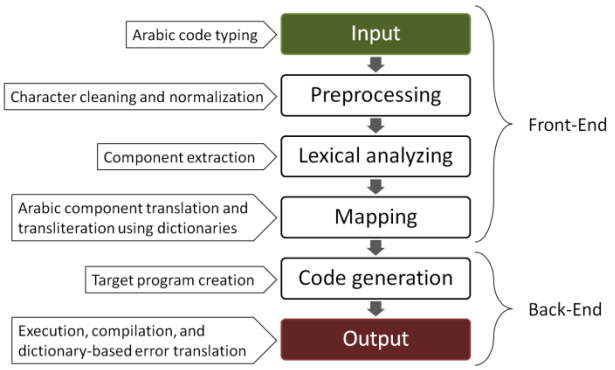

Figure 1. Implemented ArPy model

In the following steps, we will describe each block of the implemented ArPy model in details:

A. *Arabic source code (Input)*

Our method enables Arabic-speaking users to write code using Arabic keywords, adhering to Python's basic programming rules.

B. *Pre-processing*

For effective Arabic code processing, the preprocessing step is crucial to reduce the noise for the next steps. We filter out diacritics and useless spaces and we unify Arabic digits and punctuation.

C. *Lexical analysis*

The method matches simple and efficient *regex* patterns to accurately segment the preprocessed code into components. Thus, all components are detected and extracted with their types (keywords, identifiers, symbols, and numbers).

D. *Arabic word mapping*

We perform flexible mapping of Arabic components to Python equivalents, accommodating multiple variants. Unmatched keyword and identifier components are transliterated. Figure 2 presents some elements in our components dictionary, displaying Arabic words with their Python equivalents.

E. *Code generation*

In this step, we iterate and assemble the extracted components in order to generate a valid and functional Python code with preserved indentations.

F. *Result (Output)*

The final step consists of executing the Python code and showing the result in the output interface. Our solution assists Arabic learners by detecting and displaying errors in the generated code. We have generated a dictionary to translate each error category into Arabic. If any error occurs during executing the code, the translation and line position of the error are shown in the output zone. Users can conveniently edit the Arabic code within the same interface. Figure 3 presents some entries from our errors dictionary, displaying error categories with their Arabic translations.

```
# Arabic word mapping dict
DICT_AR_KEYWORDS = {
    "ادخل": "input",
    "ادخال": "input",
    "والا اذا كان": "elif",
    "او اذا": "elif",
    "اذا": "if",
    "اذا كان": "if",
    "والا": "else",
    "غير ذلك": "else",
    "طالما": "while",
    "لكل": "for",
    "استمر": "continue",
    "واصل": "continue",
    "اكسر": "break",
    "اكسر الحلقة": "break",
    "في": "in",
    "مرر": "pass",
    "تجاهل": "pass",
    "داله": "def",
    "رقم صحيح": "int",
    "تساخ": "=",
    "ياخذ": "=",
    "تساوي": "==",
    "يساوي": "==",
    "اكبر من": ">",
    "اصغر من": "<",
    "و": "and",
    "او": "or",
```

Figure 2. Sample entries of dictionary for mapping stage

```
# Dict to translate Python error types
ERROR_TO_ARABIC = {
    "AssertionError": "خطأ في التأكيد",
    "AttributeError": "خطأ في السمة",
    "EOFError": "خطأ في نهاية الملف",
    "FloatingPointError": "خطأ في الأعداد العشرية",
    "GeneratorExit": "انتهاء المولد",
    "ImportError": "خطأ في الاستيراد",
    "IndexError": "خطأ في الفهرس",
    "KeyError": "خطأ في المفتاح",
    "KeyboardInterrupt": "إلغاء بواسطة لوحة المفاتيح",
    "MemoryError": "خطأ في الذاكرة",
    "NameError": "خطأ في الاسم",
    "NotImplementedError": "خطأ غير مُنفذ",
    "OSError": "خطأ في النظام",
    "OverflowError": "تجاوز الحد الأقصى للقيمة",
    "ReferenceError": "خطأ في الإشارة",
    "RuntimeError": "خطأ أثناء التشغيل",
    "StopIteration": "توقف التكرار",
    "SyntaxError": "خطأ في الصياغة",
    "IndentationError": "خطأ في التنسيق",
    "TabError": "خطأ في التبويب",
    "SystemError": "خطأ في النظام",
    "SystemExit": "خروج من النظام",
    "TypeError": "خطأ في النوع",
    "UnboundLocalError": "خطأ في المتغير المحلي غير المُحدد",
    "UnicodeError": "خطأ في الترميز",
    "UnicodeEncodeError": "خطأ في ترميز اليونيكود",
    "UnicodeDecodeError": "خطأ في فك ترميز اليونيكود",
    "UnicodeTranslateError": "خطأ في ترجمة اليونيكود",
    "ValueError": "خطأ في القيمة",
    "ZeroDivisionError": "خطأ في القسمة على الصفر"
}
```

Figure 3. Dictionary used for errors translation

IV. RESULTS

The results obtained from the experiments conducted in this research will provide important insights into the effectiveness and efficiency of the implemented method through the evaluation of correctness of generated codes, execution time, examples of treated samples, and feedbacks from novice participants. Although no state-of-the-art method is available, the dataset and results can serve as a reference for future studies. The results evaluate the proposed method's performance and suitability for the targeted task.

A. Experimental setup

The experiments were conducted on a laptop computer running Windows 10 with an Intel Core i5-9300H CPU clocked at 2.40 GHz and 8 GB of RAM. The computer was equipped with a GIGABYTE NVMe SSD with 512 GB of storage, which was used for all data storage and retrieval. The whole solution was implemented using Python 3.7 leveraging its computational logic capabilities for implementing regular expressions and dictionaries. Moreover, the Flask module was used for the web interface.

B. Dataset

We have evaluated the performance of ArPy by using 4,999 snippets of Python codes from the dataset constructed by Chowdary [19]. To ensure the reliability of our experimental results, we identified and corrected several inconsistencies and errors in the formatting of this dataset, such as wrong indentations, misplaced snippet separators, and corrupted keywords, etc. Knowing that all the codes were written in Python language and there is no available dataset for Arabic Python programming language which fit our requirements. Thus, we translated all the codes into Arabic. To perform this with less manual effort, we selected the unique English Python keywords and replaced them by equivalent words in Arabic. Also, we translated and transliterated other unique English tokens to Arabic. In addition, we slightly and randomly edited some of the resulting Arabic words to simulate the different human Arabic writing styles. The information of the used dataset of Python code snippets is shown in Table I. Since there are currently no existing methods specifically designed for Arabic Python programming, researchers can request our dataset to enable comparisons of different methods in future studies.

TABLE I. INFORMATION ABOUT ARABIC AND ENGLISH DATASETS

Unit	Number in original English codes	Number in translated Arabic codes
Characters	1,128,001	1,176,791
Tokens	311,838	346,606
Lines	42,745	
Snippets	4,999	

C. Metrics

The evaluation of the implemented ArPy will be conducted using several metrics. One of the main metrics is edit similarity, which will be used to assess the correctness of the translation generated by the model. We will use this metric for both character-level and token-level comparisons. The character-level edit distance metric measures the distance between two text samples by counting the number of character insertions, deletions, and substitutions needed to transform one sample into the other. We follow the same principle for token-level but using a sequence of tokens instead of sequence of characters. We need to transform the distance value into similarity value which reflects the degree of similarity between the generated English Python code from the Arabic code and the target, or the gold standard, English Python code selected from the dataset. To transform the edit distance value into a similarity which varies between 0% and 100%, we used the following formula [20]:

$$sim(tar, gen) = 1 - \frac{edit\_distance(gen,tar)}{max(|gen|,|tar|)} \quad (1)$$

where *tar* is the target English Python code in the dataset from which the Arabic code is written, and *gen* is the English Python code generated by our method from the Arabic code. *edit_distance* is the character-level or the token-level edit distance between the texts of the two English Python codes. |*tar*| and |*gen*| are the number of characters in the target and the generated codes, respectively, while *max* is the maximum operator. For a given set of snippets, we will calculate the edit similarity for each pair of snippets consisting of the original (target) snippet and the generated snippet, and then take the average of all the results. This will provide us with the overall average edit similarity *AES* for the set of snippets. For that, we use the following formula [20]:

$$AES = \frac{1}{n}\sum_{i=1}^{n} sim(tar_i, gen_i) \quad (2)$$

Where *n* is the number of snippets in the dataset, tar_i is the i^{th} original (target) snippet, gen_i is the i^{th} generated snippet, and $sim(tar_i, gen_i)$ is the edit similarity between the i^{th} original snippet and the i^{th} generated snippet.

We will also use other metrics to evaluate the model's performance. For instance, we will measure the execution time of the model to evaluate its efficiency. We have also provided input and output examples which illustrate the performance of our model and demonstrate how our model translates English code into Arabic. Additionally, we will analyze the feedback provided by the users to gain insights into the quality of the translations.

D. Evaluation of Translated Code Correctness

In Table II, we compared the original English code with the generated English code by ArPy from its Arabic translation, excluding several parts. The similarity achieved was about 84% at the character level and about 87% at the token-level, effectively preserving code functionality with good accuracy.

TABLE II. DATASETS EDIT-SIMILARITY COMPARISON

Excluded parts	Character-level similarity	Token-level similarity
No exclusions	84.20%	87.96%
Spaces excluded	83.40%	87.96%

Excluded parts	Character-level similarity	Token-level similarity
Strings excluded	83.78%	88.03%
Tokens anonymized	92.08%	87.12%
Logical comparisons	92.73%	88.76%

E. Execution time

To evaluate the performance of ArPy, we measured the model's execution time on a range of input sizes. Table III shows the execution times for input code sizes ranging from 10 to 4,999 (all) snippets. Additionally, we provided information about the number of lines, characters, and tokens in each range of snippets. This information is presented in separate columns in Table III.

TABLE III. EXECUTION TIME FOR DIFFERENT CODE SNIPPET RANGES

Snippets	Lines	Characters	Tokens	Execution time (milliseconds)
10	104	2,187	652	3.99
100	752	19,023	5,698	29.00
250	1,626	45,846	13,503	70.99
500	3,641	105, 187	30,512	169.00
750	6,688	183,902	53,526	357.02
1,000	8,885	246,966	72,242	412.09
2,000	17,142	461,515	134,525	775.95
3,000	25,002	684,332	199,420	1,192.90
4,000	34,215	935,255	274,445	1,652.63
4,999	42,745	1,176,582	346,639	1,993.09

Figure 4 depicts a curve that represents the relationship between the snippet code sizes with the execution time. The x-axis corresponds to the code ranges, while the y-axis represents the execution time.

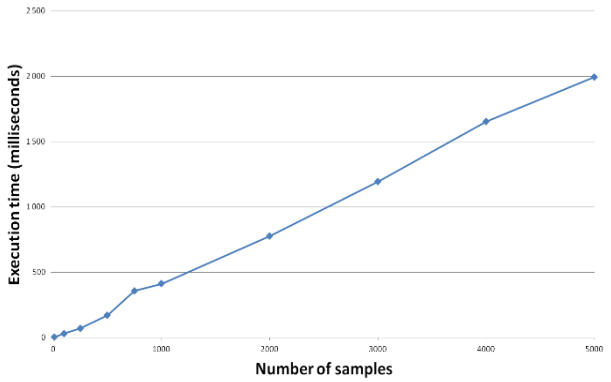

Figure 4. Execution time for different ranges of snippets

Overall, ArPy performs very quickly and adds negligible time to the execution of normal Python code. For beginner programmers, the difference in speed is not even noticeable, as it is less than 4 milliseconds for 10 snippets. However, since Python is an interpreted language and is not compiled into low-level machine language, the overall method may be slower than other programming languages such as C++.

Figures 5 and 6 illustrate snapshots for two samples of codes using the implemented ArPy GUI package. ArPy has maintained good accuracy even with challenging cases, and successfully mapping Arabic Python keywords into executable instructions. Results and errors are shown in the output zone. However, complex strings and some uncovered standard identifiers may lead to some errors.

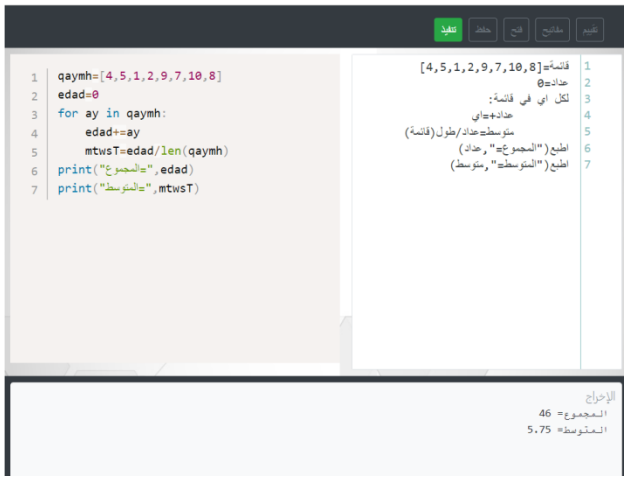

Figure 5. Sample 1: ArPy code with correct output

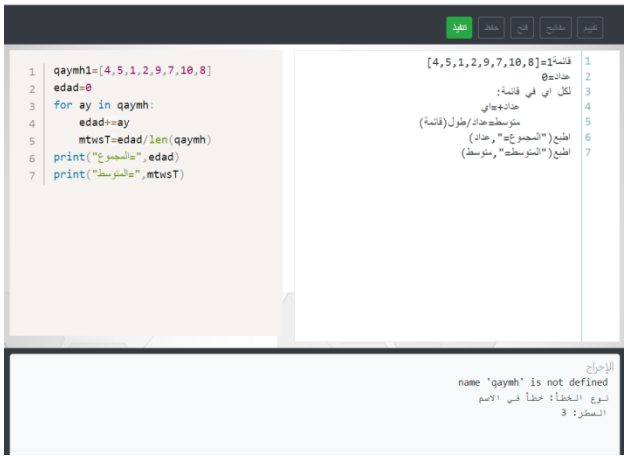

Figure 6. Sample 2: ArPy with output error handling

F. Feedbacks

We conducted a survey about Arpy package with 50 IT novices at the faculty of IT, Misurata using a questionnaire of 5 questions with the following feedback results, as shown in Figure 7:

- **Ease of Use:** 60% found our method very easy or moderately easy to use.
- **Effectiveness:** 78% rated the method highly effective in generating accurate code.
- **Relevance to Programming Education:** Approximately 80% found it highly relevant.
- **Task Completion Speed:** 44% found it challenging.
- **Capability for Bigger Projects:** Around 70% indicated it is not suitable for larger projects.

These findings highlight the ArPy's effectiveness and relevance, as well as areas for improvement. Further research is needed for larger projects. Also, users suggested improving the input interface.

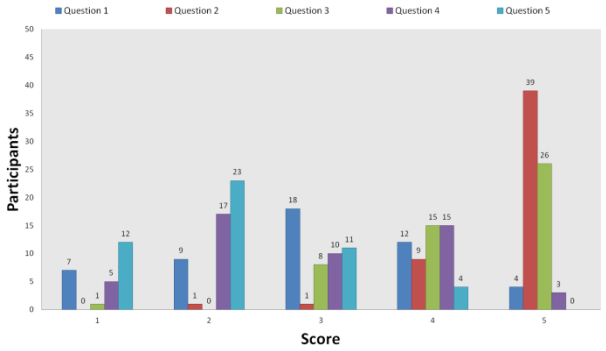

Figure 7. Responses across Feedback questions.

V. CONCLUSIONS

This research contributes to the field of natural language processing for programming languages and strives to bridge the language barrier in the field of programming education in Arabic countries.

In this paper, we have endeavored to make programming language education more accessible for novice Arabic speakers by introducing a Python programming language interface tailored to Arabic language. The main goal was to create a flexible interface for diverse writing styles, accurate Arabic code translation, and simplified debugging with Arabic error messages. Our results clearly indicate that the implemented model (ArPy) is able to efficiently generate functional English Python code from Arabic code with a similarity accuracy of about 87%. Furthermore, ArPy received positive feedbacks from the target users. As future work, we aim to enhance the user experience needed for larger projects and refine rules for more accuracy. Moreover, we will explore direct low-level compilation for faster Arabic code execution.

REFERENCES

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Addison Wesley, 2nd edition, 2006.

[2] Bashir S Abubakar, Abdulkadir Ahmad, Muktar M Aliyu, Muhammad M Ahmad, and Hafizu U Uba. An overview of compiler construction. International Research Journal of Engineering and Technology (IRJET), 2021.

[3] MT Singh. How to teach programming languages to novice student and problems in learning of students. Journal of Computing Technologies (JCT), 1(2):5, 2012.

[4] Ibrahim Nnass, Michael A Cowling, and Roger Hadgraft. Identifying the difficulties of learning programming for non-English speakers at CQUniversity and Sebha University. Journal of Pure and Applied Sciences, 21(4):290–295, 2022.

[5] Mrwan Ben Idris and Hany Ammar. The correlation between Arabic student's English proficiency and their computer programming ability at the university level. International Journal of Managing Public Sector Information and Communication Technologies, 9:01–10, 03 2018.

[6] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M Patel, and Richard Halverson. What do students feel about learning programming using both English and their native language? In 2017 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), pages 1–8. IEEE, 2017.

[7] Mrwan Ben Idris and Hany Ammar. The correlation between Arabic student's English proficiency and their computer programming ability at the university level. International Journal of Managing Public Sector Information and Communication Technologies (IJMPICT) Vol, 9, 2018.

[8] Alaaeddin Swidan and Felienne Hermans. A framework for the localization of programming languages. Available at SSRN 4385792, 2023.

[9] Hussam Hatem Abdul Razaq, Ayedh Shahadha Gaser, Mazin Abed Mohammed, Esam Taha Yassen, Salama A Mostafad, Subhi RM Zeebaree, Dheyaa Ahmed Ibrahim, Mohd Khanapi Abd Ghania, and Rabah N Farhan. Designing and implementing an Arabic programming language for teaching pupils. Journal of Southwest Jiaotong University, 54(3), 2019.

[10] Majdi Beseiso, Abdul Rahim Ahmad, and Roslan Ismail. A survey of Arabic language support in semantic web. International Journal of Computer Applications, 9(1):35–40, 2010.

[11] Hanan Elazhary. Facile programming. Int. Arab J. Inf. Technol., 9(3):256–261, 2012.

[12] Youssef Bassil and Aziz Barbar. Myprolang - my programming language: A template-driven automatic natural programming language. Lecture Notes in Engineering and Computer Science, 2173, 04 2012.

[13] Mohamed Tahar Ben Othman. Arabic computer programming education tool. International Journal on Islamic Applications in Computer Science and Technology, 4(1):25, 2016.

[14] Youssef Bassil. Phoenix–the Arabic object-oriented programming language. International Journal of Computer Trends and Technology, 67(2):7–11, 2019.

[15] Tahani Almanie, Shorog Alqahtani, Albatoul Almuhanna, Shatha Almokali, Shaima Guediri and Reem Alsofayan. Let's Code: A kid-friendly interactive application designed to teach Arabic-speaking children text-based programming. International Journal of Advanced Computer Science and Applications, 10(7), 2019.

[16] Krishan Kumar and Sonal Dahiya. Programming languages: A survey. International Journal on Recent and Innovation Trends in Computing and Communication, 5(5):307–313, 2017.

[17] Nadia Mahmood Hussien and Yasmin Mohialdeen. A Pyarabic Python library to create Arabic applications. Webology, 19:287, 10 2022.

[18] Daro C Arias Jaco and Melvin A Molina Sandoval. Design of a prototype programming language in Spanish and its compiler through tools JFlex and Java cup. In 2016 IEEE Central America and Panama Student Conference (CONESCAPAN), pages 1–6. IEEE, 2016.

[19] Pranith Chowdary. python-150k-code, 2023. URL https://www.kaggle. com/ds/2917090. Accessed: 16/10/2023.

[20] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. Journal of the ACM (JACM), 21(1):168–173, 1974.