UniversalPython - A Multilingual, Pythonic Programming Language*

* Note: This draft was submitted in partial fulfillment of the requirements for the course "Research Methodology" at FAST-NUCES

Saad Ahmed Bazaz
Software Engineering and Product Design
Grayhat
Islamabad, Pakistan
bazaz@grayhat.studio

***Abstract*—All widely used and useful programming languages have a common problem. They restrict entry on the basis of knowledge of the English language. The lack of knowledge of English poses a major hurdle to many newcomers who do not have the resources, in terms of time and money, to learn the language. Furthermore, studies back up the fact that learning is better when it's done in the person's local language. Therefore, we propose a language wrapper built on top of the Python programming language which can be directly used in the native Urdu language. This eliminates the need for any intermediate language as well. In the future, we aim to scale the language to encapsulate more languages to increase the availability of programming.**

***Index Terms*—programming language, internationalization, python, artificial intelligence, machine translation**

I. Introduction

The first step to better understanding is communication. We use language to communicate concepts and produce thoughts. When teaching and educating, it is preferable to speak in the learner's native language to facilitate the understanding process.

Computer science is a field dominated by the English language. Beginners must be acquainted with English to understand basic logical constructs. Without a real-life translator, learning programming concepts just by looking at English words is often impossible.

Therefore, there is a need for formal translation of programming constructs into other languages and a usable framework for learners to program in their native language.

II. Related Work

Studies have shown that students face problems learning the syntax and rules of a programming language initially [1]. However, learning in a local language has a significant positive impact on learning outcomes [2]. This supports the idea that programming should not be restricted by a language barrier.

A. Multilingual/Localized Programming Languages

PsueToPy [3] proposes an intermediate pseudo language to communicate with the Python interpreter. This approach adds complexity for users due to potential inconsistencies in machine translation. Additionally, PsueToPy lacks a standard structure, making it difficult to transition between languages.

An African-based programming language [5] faced scalability issues and a lack of community backing. Another approach involved editing the Python source code to accommodate Chinese symbols [6]. However, this method is difficult to implement and maintain across different versions and languages.

Several attempts have been made to create Urdu programming languages [7, 8, 9, 10]. However, they suffer from issues like using Roman Urdu instead of Arabic script, lacking open-source libraries, and limited scalability to other languages.

UrduScript [7] is similar to our approach but uses JavaScript instead of Python and Roman Urdu script. Our method leverages Python's extensive libraries and utilizes the more familiar Arabic script for Urdu speakers.

B. Multilingual Programming Environments

Approaches have been made to address the issue of development environments not supporting multiple languages [11, 12]. Additionally, some work translates Python programs into traditional Chinese for explanations [13]. However, this method uses Google Translate, making it slow and unsuitable for programming purposes.

Our framework allows easy support for UniversalPython plugins in existing Python IDEs. It acts as a bridge, translating Urdu code into English before passing it to the IDE.

III. Approach

A. Making the Language

We propose a framework as shown in "Fig. 1", which is a wrapper around the Python Engine. The user writes code in Urdu, for example:

This is passed to "UniversalPython", which first translates the code to English using Lexical Analysis and

Parsing with the PLY library. To do this, it first loads the Urdu dictionary, which is a YAML file containing mappings from Urdu to English. This dictionary is a mapping of each Urdu word to a Python keyword. Here is an example of such a dictionary: In PLY, we have the option to reserve some keywords so that the library automatically tokenizes them. We set a Grammar Rule which, whenever a reserved keyword (i.e. a word which is present as a key in a language dictionary) is tokenized, simply looks for the token in the language dictionary (key) and replaces it with the corresponding English keyword (value). We also set a Grammar Rule to ignore all content within double quotes or single quotes (i.e. strings and docstring) and to ignore content in comments (which start with a #). We achieve this using Regular Expressions. Urdu numbers also lie on the Unicode scale. (or roman 0) is at 1776, while (or roman 9) is at 1785.

Keeping this in mind, we define a Regular Expression which detects any symbols equal to or between this range, 1776 - 1785. These are the Urdu digits. We then use the same language dictionary to translate these numbers into roman numerals. For example, becomes 5, becomes 90, becomes 10, becomes 2022. Furthermore, we also replace all periods (.) and commas (,), as these look different in Urdu than they do in English.

The rest of the code remains untouched. Whether it is a symbol,like :, (or), etc, or even if it causes the lexer to crash, we ignore it so we can preserve the original structure of the code as much as possible. It is not required to translate such symbols and errors anyway: They are meant to be handled by Python, not UniversalPython. Back to our initial example code, our engine detects and replaces it with print , is replaced with if , is replaced with else , and all Arabic digits are replaced with Roman digits. Hence the translated code would be:

The above code is essentially vanilla Python code which can now simply be executed by the Python engine. The Urdu variable name does not present any issue to Python, as from Python 3.0 onwards, Unicode is fully supported. So the above code is passed on to the Python engine. The Python engine outputs some response; it can be some print statements which the user entered, compiler/interpreter warnings, and/or errors. This response is passed up to UniversalPython, where again it is tokenized to replace keywords in case of error messages. In our example, since there are no errors, it simply outputs the response as-is. So the response would be:

To demonstrate the ease by which plugins can be made for UniversalPython, we made a wrapper for the IPython kernel in which we imported UniversalPython as a package, and processed the code (i.e. translated it from Urdu to English) before it was passed onto IPython. This way, we overrode the

functions and achieved a working kernel for UniversalPython. It works line-by-line while maintaining the program memory. This also gave us a visual interface for the language to test it thoroughly.

The Urdu dictionary is a YAML file containing mappings from Urdu to English keywords, is used for translation. The PLY library allows reserving keywords for automatic tokenization and replacement with their English equivalents. Additionally, grammar rules are set to ignore content within quotes, comments, and Urdu numbers (which lie on the Unicode scale).

IV. Experimentation

We propose the following requirements which UniversalPython should meet in order than it is considered effective:

1) Programs which work in Python, should be recreatable in UniversalPython, and vice versa.

2) UniversalPython should operate as a reasonable speed which at least does not disturb the programmer.

3) Converting Plugins made for Python, into Urdu Python, should be fairly easy.

A. Benchmarks with Python

Time: Check the execution time / performance of Python vs UniversalPython. We use a benchmarking tool called hyperfine which runs each program multiple times to produce a mean execution time.

Conversion: Convert simple programs from English Python to UniversalPython, and test if they still work. Our testing mechanism for the above is as below: 1) We take an existing Python program, lets say multiplication.py. 2) Run the UniversalPython system in reverse. This will flip the language dictionary and generate an Urdu version of the program (all the keywords would be translated from English to Urdu). Save it as multiplication.ur.py. 3) Run multiplication.ur.py using UniversalPython, and multiplication.py using Python. If both give the same output on the terminal then no data loss has occurred; Both Urdu program and English program output the same message. Hence we can say that the program has safely converted from English to UniversalPython and vice versa without breaking the code or changing the logic.

We take simple algorithmic programs from TheAlgorithms/Python [14], a repository containing implementations of well-known algorithms, in the Python programming language. We run a loop over them and run the above algorithm on each to test.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly

state the units for each quantity that you use in an equation.

- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Discussion and Future Work

UniversalPython can be developed as an extensible programming language which would grow alongside Python and would be interoperable with it. It is also maintainable, because since it is simply a wrapper, only the keywords need to be updated in the dictionary, and it will work

for any future (or past) versions of Python. Furthermore, it is scalable to other languages because the dictionary structure allows for any language to be added to UniversalPython.

To improve the current implementation, it would be useful to accommodate as many Python keywords as possible, preferably all of them. It would also be useful to separately have translations for major libraries/packages such as pandas, numpy, cv2, etc. A scalable and efficient method for this would be to automatically scan such libraries(the Python source code) to identify important keywords, automatically translate them to the nearest and best possible Urdu word, and add them to the Grammar. This can be achieved using Natural Language Processing, Deep Learning, Machine Learning and similar methods.

It is possible to make this framework generic. We managed to make a dictionary for Hindi Python, and it works just as well as UniversalPython. We would prefer support for more languages such as Pashto, Punjabi, Balochi, Sindhi, Kashmiri and more.

In the current framework, making it possible for the user to choose whether or not to detect variables and functions, and machine-translate them into their corresponding names, could make the experience better.

It would also be helpful to conduct User Experience tests for UniversalPython, so that it can be determined whether or not UniversalPython really does benefit people who speak Urdu to learn programming.

Designing and developing a better test metric to test the effectiveness of such a framework would also be useful for UniversalPython.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 1", even at the beginning of a sentence.

Figure 1: Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.

Table I: A application of the framework to a set of programming languages

Aspect/Prog Language	UniversalPython	Scratch	Snap!	Dolittle	Ebda3	Qalb	Wenyan	Excel	PSeInt	Rapture	H
Language	Multi	Multi	Multi	Japanese	Arabic	Arabic	Chinese	Multi	Spanish	Russian	M
Alignment	N	N	N	N	N	T	N	T	N	T	N
Non-English keywords	●	●	●	●	●	●	●	●	●	●	
Non-Latin variable names	●	●	●	●	●	●	●	○	●	●	
Non-English productions	●	○	●	-	-	●	○	●	-	-	
Non-English numerals	●	-	-	-	-	●	●	●	-	-	
Characters without meaning	●	-	-	-	-	●	●	-	-	-	
Diacritics	●	●	●	○	○	●	○	-	●	○	
Alternative keywords	●	-	-	-	○	-	●	-	●	○	
Localized punctuation	●	○	○	○	○	○	●	●	-	-	
Right to left support	●	○	-	○	●	●	○	●	-	-	
Multi-lingual programming	●	-	-	-	-	-	-	-	-	●	
Error messages	-	○	○	●	○	○	-	●	●	-	
Multi-lingual 3rd-party libraries	○	○	-	○	○	○	○	●	-	-	

Table II: Table Type Styles

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.