# Project: HybridPython - A Multilingual Programming Alternative

### Objective:

Create a versatile hybrid programming language, "HybridPython," that bridges the gap between different human languages and programming constructs. The target is to design an alternative that includes localized keywords and syntax, allowing speakers of diverse languages to program effectively. This project aims to connect with over 364 million people worldwide whose native languages are not primarily represented in traditional programming languages.

### Supported Languages and Speaker Statistics:

1. **Bangla (Bengali)**

Approximately **230 million speakers**

**Reference**: Ethnologue (2021) - Bengali

1. **Turkish**

Approximately **75 million speakers**

**Reference**: Ethnologue (2021) - Turkish

1. **Hindi**

Approximately **600 million speakers** (including second-language speakers)

**Reference**: Ethnologue (2021) - Hindi

1. **Japanese**

Approximately **125 million speakers**

**Reference**: Ethnologue (2021) - Japanese

1. **Spanish**

Approximately **580 million speakers** (including native and second-language speakers)

**Reference**: Ethnologue (2021) - Spanish

1. **Russian**

Approximately **258 million speakers**

**Reference**: Ethnologue (2021) - Russian

1. **Arabic**

Approximately **310 million speakers** (varies widely due to dialects)

**Reference**: Ethnologue (2021) - Arabic

1. **Chinese (Mandarin)**

Approximately **1.1 billion speakers** (including all dialects, but Mandarin is the most spoken)

**Reference**: Ethnologue (2021) - Chinese

1. **Urdu**

Approximately **170 million speakers**

**Reference**: Ethnologue (2021) - Urdu

1. **Sinhala**

Approximately **16 million speakers**

**Reference**: Ethnologue (2021) - Sinhala

### Features of HybridPython:

### ****Localized Keywords:****

Programming keywords and functions will be available in multiple languages (e.g., if in English, যদি in Bengali, and equivalent terms in other languages).

**Multi-Syntax Compatibility:**

Users can write code using syntax in their native language or mix multiple languages for a "hybrid" approach.

**Universal Compilation:**

A single compiler/interpreter that can parse multilingual syntax and execute code seamlessly.

**Intelligent Mapping:**

Automated translation of code written in one language to another.

Example: Code written in Hindi will be translatable to its English equivalent for universal understanding.

**Integration with Global Languages:**

Support for a total of **10 major languages** and **3+ sub-languages/dialects**.

**Educational Support:**

Built-in tutorials in all supported languages to make programming accessible to everyone.

### Example Code:

#### Python Code in English:

if age > 18:

print("You are an adult.")

else:

print("You are a minor.")

#### The Same Code in Bengali (Bangla):

যদি বয়স > 18:

মুদ্রণ("আপনি একজন প্রাপ্তবয়স্ক।")

অন্যথা:

মুদ্রণ("আপনি একজন নাবালক।")

### Goals:

**Cultural Inclusivity:**

Break language barriers in programming.

Enable non-English speakers to learn coding more effectively.

**Hybrid Connectivity:**

Provide a "hybrid" approach where programming in multiple languages is possible within the same file.

**Target Audience:**

Primary focus on speakers of Bangla, Hindi, Urdu, Sinhala, Turkish, Arabic, Chinese, Japanese, Spanish, and Russian.

Expand support to sub-languages and dialects based on community demand.

**Adoption of Multilingual Programming:**

Target to connect with over **364 million native speakers** across supported languages.

### Implementation Strategy:

**Research and Development:**

Study programming needs in non-English-speaking regions.

Collaborate with linguists and educators to develop accurate translations of programming terms.

**Compiler Development:**

Build an interpreter capable of recognizing multilingual syntax.

Ensure backward compatibility with standard Python.

**Community Involvement:**

Open-source the project to encourage contributions.

Create forums and resources for feedback and development.

**Educational Outreach:**

Partner with schools and institutions to promote multilingual programming.

Develop courses and certifications in HybridPython.

### Vision: B.1.12.29.0V

HybridPython will revolutionize programming by making it accessible to millions of

In Hybrid-Python, the built-in functions and keywords allow users to perform a wide range of operations, from mathematical calculations to object manipulation. These features enhance Python's versatility and ease of use for both beginners and experienced developers.

1. abs(): Returns the absolute value of a number.
2. all(): Returns True if all elements in an iterable are true; otherwise, returns False.
3. any(): Returns True if any element in an iterable is true; otherwise, returns False.
4. bin(): Converts an integer to a binary string prefixed with '0b'.
5. bool(): Converts a value to a Boolean (True or False).
6. bytearray(): Returns a new bytearray object, which is a mutable sequence of bytes.
7. bytes(): Returns an immutable sequence of bytes.
8. callable(): Checks if an object appears callable (i.e., if it can be called as a function).
9. chr(): Converts an integer to its corresponding character.
10. classmethod(): Converts a method into a class method.
11. compile(): Compiles a source into a code object that can be executed.
12. complex(): Creates a complex number from real and imaginary parts.
13. delattr(): Deletes an attribute from an object.
14. dict(): Creates a dictionary.
15. dir(): Returns a list of attributes and methods of an object.
16. divmod(): Returns the quotient and remainder when dividing two numbers.
17. del: Deletes a reference to an object.
18. elif: A conditional statement used after an if statement, evaluating another condition if the previous condition was false.
19. else: Executes a block of code if the preceding if or elif condition fails.
20. enumerate(): Adds a counter to an iterable and returns it as an enumerate object.
21. eval(): Executes a string containing Python code.
22. exec(): Executes a dynamically created Python program (string or code object).
23. filter(): Filters an iterable by applying a function that evaluates to True or False.
24. float(): Converts a value to a floating-point number.
25. format(): Formats a string by replacing placeholders with values.
26. frozenset(): Returns an immutable set.
27. getattr(): Returns the value of an object's attribute.
28. globals(): Returns a dictionary representing the global symbol table.
29. hasattr(): Checks if an object has a specified attribute.
30. hash(): Returns the hash value of an object.
31. help(): Displays help information about an object.
32. id(): Returns the identity of an object.
33. input(): Reads a line of input from the user.
34. int(): Converts a value to an integer.
35. isinstance(): Checks if an object is an instance of a specified class.
36. issubclass(): Checks if a class is a subclass of another class.
37. join(): Joins elements of an iterable into a single string.
38. len(): Returns the length of an object.
39. list(): Creates a list from an iterable.
40. locals(): Returns a dictionary representing the local symbol table.
41. map(): Applies a function to all items in an iterable.
42. max(): Returns the largest item from an iterable.
43. memoryview(): Creates a memory view object from a byte-like object.
44. min(): Returns the smallest item from an iterable.
45. next(): Retrieves the next item from an iterator.
46. object(): Returns a new featureless object.
47. oct(): Converts an integer to its octal representation as a string.
48. open(): Opens a file and returns a file object.
49. ord(): Converts a character to its corresponding integer value (ASCII or Unicode).
50. pass: A placeholder statement used when no action is needed.
51. pow(): Returns the power of a number, optionally with a modulus.
52. print(): Outputs text or objects to the console.
53. property(): Returns a property descriptor for an attribute.
54. range(): Returns a sequence of numbers within a specified range.
55. repr(): Returns a string that represents the object.
56. reversed(): Returns a reversed iterator of an object.
57. round(): Rounds a number to a specified precision.
58. set(): Creates a set from an iterable.
59. setattr(): Sets the value of an object's attribute.
60. slice(): Returns a slice object used for slicing sequences.
61. sorted(): Returns a sorted list from an iterable.
62. staticmethod(): Converts a method into a static method.
63. str(): Converts a value to a string.
64. sum(): Sums the elements of an iterable.
65. tuple(): Creates a tuple from an iterable.
66. type(): Returns the type of an object.
67. upper(): Converts a string to uppercase.
68. vars(): Returns the \_\_dict\_\_ attribute of an object, representing its attributes.
69. while: A loop that repeatedly executes as long as a condition is true.
70. with: Used to wrap the execution of a block of code, ensuring that resources are properly managed (e.g., closing files).
71. xor(): Performs a bitwise XOR operation on two numbers.
72. yield: Used in functions to return a generator, allowing iteration over values lazily.
73. zip(): Combines multiple iterables element-wise into tuples.

#### Other Keywords

1. if: Begins a conditional statement.
2. else: Executes when the if condition is false.
3. elif: Provides an alternative condition to evaluate if the if condition is false.
4. try: Starts a block of code that may raise exceptions.
5. except: Catches and handles exceptions raised in the try block.
6. finally: Executes code after the try and except blocks, whether an exception was raised or not.
7. raise: Raises an exception explicitly.
8. lambda: Creates an anonymous function.
9. global: Declares a variable as global, allowing it to be accessed across functions.
10. nonlocal: Declares a variable in the nearest enclosing scope, excluding global scope.
11. import: Imports modules or packages.
12. from: Imports specific items from a module.
13. as: Creates an alias for an imported module or item.
14. True: Represents the Boolean value True.
15. False: Represents the Boolean value False.
16. None: Represents the absence of a value or a null value.

There are different types of documentation available: developer documentation and user documentation. Below, you will find a comprehensive list of all methods, keywords, and other related elements

**Bangla (Bengali)**

****

**Turkish**

****

**Japanese**

****

**Spanish**



**Russian**

****

**Arabic**

****

**Chinese (Mandarin)**

****

**Hindi**

****

**Urdu**

****

**Sinhala**

****