THIRAN - TAMIL PROGRAMMING LANGUAGE

A PROJECT REPORT

Submitted by

REG NO	NAME
1911109	BALAPATHY V
1911147	SURYA NARAYANAN S
2011906	SARAVANAN K

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER SCIENCE AND ENGINEERING

Under the guidance of

Dr. J. NAFEESA BEGUM, M.E., Ph.D.,

Professor and Head

Department of Computer Science and Engineering

GOVERNMENT COLLEGE OF ENGINEERING

(AUTONOMOUS)

BARGUR, KRISHNAGIRI - 635 104

(Affiliated to Anna University, Accredited by NAAC with 'B' Grade)

ANNA UNIVERSITY: CHENNAI - 600 025

MAY 2023

GOVERNMENT COLLEGE OF ENGINEERING

(AUTONOMOUS)

BARGUR, KRISHNAGIRI – 635 104

Department of Computer Science and Engineering

BONAFIDE CERTIFICATE

Certified that this project titled **"THIRAN - TAMIL PROGRAMMING LANGUAGE"** is the bonafide work of **BALAPATHY V (1911147), SURYA NARAYANAN S (1911147)** and **SARAVANAN K (2011906)** who carried out the project work under my supervision.

Dr. J. NAFEESA BEGUM, M.E., Ph.D.,
SUPERVISOR,
Professor [CAS],
Department of CSE,
Govt. College of Engineering,
Bargur – 635 104

Dr. J. NAFEESA BEGUM, M.E., Ph.D.,
HEAD OF THE DEPARTMENT,
Professor [CAS],
Department of CSE,
Govt. College of Engineering,
Bargur – 635 104

Submitted for the Thesis Viva Voce Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we would like to record our deepest gratitude to **our parents** and **siblings** for their constant encouragement and support which motivated us to complete our project on time.

We express our sincere gratitude and thanks to our respected Principal **Dr. R. VIJAYAN, M.E., Ph.D.,** and our respected Head of the Department of Computer Science and Engineering **Dr. J. NAFEESA BEGUM, M.E., Ph.D.,** for giving us the opportunity to display our professional skills through this project.

We want to express our deep and wholehearted thanks to our project guide, **Dr. J. NAFEESA BEGUM, M.E., Ph.D.,** Department of Computer Science and Engineering for her expert knowledge, insightful feedback, and constant encouragement which were instrumental to this project. We are also grateful for the time she took to provide us with constructive criticism and helpful suggestions, which helped us to refine our ideas and improve our work.

We thank all **our teaching and non-teaching staff** members of the Computer Science and Engineering for their passionate support for helping us to identify our mistakes and also for the appreciation they gave us in achieving our goal.

Furthermore, we thank **Dr. MUTHAIAH ANNAMALAI** for his contributions to Tamil Computing which inspired us greatly to take up this project. Special thanks to **Mr. S. MANJUNATHAN, B.E.,** Tata Consultancy Services (TCS) for suggesting us the name “THIRAN”.

BALAPATHY V (1911109)

SURYA NARAYANAN S (1911147)

SARAVANAN K (2011906)

ABSTRACT

This project aims to develop a simple and easy-to-learn programming language called “Thiran” (Tamil: திறன்) that allows coding in Tamil. This programming language will help school students to develop their problem-solving skills by making code more readable and understandable. It will familiarize them with the concepts and fundamentals of programming before moving on to real-world programming languages. Furthermore, this project will contribute to the growth of the technology industry in Tamil-speaking regions, leading to more diverse and inclusive programming solutions.

Keywords: Tamil, Programming Language, Problem-solving, Logical thinking

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 - INTRODUCTION

1.1 MOTIVATION FOR THIRAN:

In nearly every widely used programming language, the names of symbols and keywords (e.g., for, while, if, else) are borrowed from English. Education research suggests that learning in one's local language can have a positive impact on the outcomes of learning. Accordingly, we had the motivation to develop a Tamil Programming Language called Thiran (Tamil: *திறன்*, lit. "ability") to cater to the educational needs of native Tamil-speaking K-12 students.

1.2 THIRAN PROGRAMMING LANGAUGE:

Figure 1- Thiran Logo

Thiran is an interpreted, imperative, structured, and procedural programming language that enables coding in Tamil, an official language spoken by millions worldwide. Thiran is completely free and open-source, licensed under GNU GPLv3, designed to provide a simple and easy-to-learn syntax that facilitates fast and efficient development. Thiran aims to bridge the language barrier in vernacular (Tamil) computer science education. Thiran's open-source nature also ensures that the language is free to use, distribute, and modify, making it an ideal choice for educational institutions and developers looking to promote accessibility and diversity in computer science education. Inspired by Python and Ezhil, Thiran is optimized for simplicity and ease of use while offering a comprehensive set of features.

CHAPTER 2 - EXISTING SYSTEM

2.1 EZHIL PROGRAMMING LANGUAGE:

Figure 2 – Ezhil Logo

Ezhil (எழில்) is an open-source, interpreted, procedural Tamil script based programming language. It was the first freely available programming language in the Tamil language. It was designed by **Muthiah Annamalai**. The best thing about Ezhil is that its syntax allows statements to be closer to the Subject-Object-Verb structure (**SOV**), of the Tamil language.

2.2 THIRAN VS EZHIL:

The main differences between Thiran and Ezhil are:

- Thiran focuses on simplicity over preserving Tamil sentence structure (SOV), while Ezhil focuses on preserving Tamil sentence structure (SOV).
- Though implemented in Python, Thiran is self-contained and doesn't support any Python functions or elements, whereas Ezhil is tightly integrated into Python runtime and has full access to the Python programming language environment.
- Apart from the above, just like any two programming languages, Thiran and Ezhil have their syntactical differences.

Some examples to show the syntactical difference between Thiran and Ezhil are given in the next page.

- **Decision-making statement:**

THIRAN	EZHIL

சரியெனில் (2 > 3):
    # true branch
தவறெனில்:
    # false branch
முடி
	@ (2 > 3) ஆனால்
	    # true branch
	இல்லை
	    # false branch
	முடி

Table 1 – Decision-making statements

- **While loop:**

THIRAN	EZHIL

உண்மைவரை (க < 5):
    # code to loop
முடி
	@ (க < 5) வரை
	    # code to *loop*
	முடி

Table 2 – while loop

- **Print statement:**

THIRAN	EZHIL
காட்டு ("வணக்கம்!")	பதிப்பி "வணக்கம்!"

Table 3 – print statement

CHAPTER 3 - THIRAN IMPLEMENTAION

Thiran, just like any other programming language, has a set of rules and instructions (called the Grammar of the language) that govern how programmers can create programs using it. The process of translating these rules and instructions into a language that computers can understand is accomplished through a series of steps:

1. Lexical analysis (also known as tokenization)
2. Parsing
3. Interpretation
4. Execution

3.1 LEXICAL ANALYSIS:

The first step in this process is lexical analysis. It is done through a program called ThiranLexer.py. The lexer reads the source code and breaks it down into a stream of tokens, which are essentially the building blocks of the language. Tokens can be things like keywords, operators, and identifiers. The lexer then passes these tokens to the parser.

3.2 PARSING:

The next step in the process is parsing. Parsing is done through a program called ThiranParser.py. The type of parser used is recursive descent parser. The parser takes the stream of tokens generated by the lexer and creates a tree structure known as an abstract syntax tree (AST) using the Grammar of the language (refer to the annexure of this documentation). The AST represents the program's structure and can be thought of as a blueprint for the program's execution. The parser checks if the syntax of the program is correct and generates error messages if it encounters any problems.

3.3 INTERPRETATION:

The next step is interpretation. Interpretation is done through a program called ThiranInterpreter.py. The interpreter reads the AST generated by the parser and executes the instructions it contains. The interpreter works by walking through the AST, node by node, and performing the operations specified in each node. This is different from a compiler, which translates the program into machine code that can be executed directly by the computer.

3.4 EXECUTION:

Finally, the program is executed. During this stage, the interpreter program reads the instructions generated during the previous stages and carries out the corresponding actions using the Python programming language. This may include reading and writing data, performing calculations, and displaying output. Once the program has completed its execution, the interpreter program terminates.

The figure below shows how Thiran works internally.

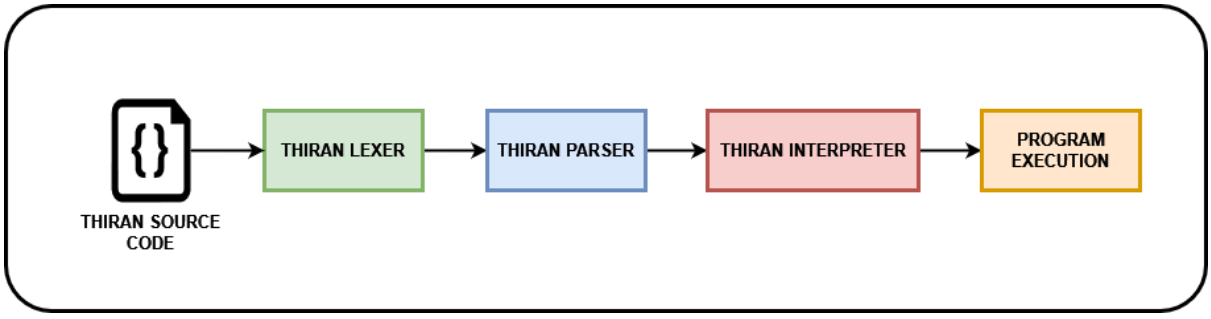

Figure 3 – How Thiran works internally

3.5 THIRAN SYSTEM IMPLEMENTATION:

Thiran was coded entirely in an object-oriented fashion using the Python programming language. Thiran converts the source program code into Python AST node objects using the language's Grammar and executes them. Also, since

this programming language revolves around the usage of Tamil language, the standard character encoding is set to ‘UTF-8’.

The entire code is licensed under GNU GPLv3. To view or download the entire code behind Thiran, refer to the annexure of this documentation.

The modules (code files) behind Thiran are:

- Thiran.py
- ThiranLexer.py
- ThiranParser.py
- ThiranInterpreter.py
- Error.py
- Globals.py
- ThiranImport.py
- ThiranBuiltIns.py

3.6 ADDING NEW FEATURES OR UPDATING THIRAN:

Since Thiran is licensed under GNU GPLv3, anyone can contribute to its development, which as always, is highly appreciated.

To add any new features or update any existing parts of Thiran, the following steps need to be performed:

1. Update the Grammar for the language
2. Update ThiranLexer.py (to recognize the new Tokens)
3. Update ThiranParser.py (to construct AST using the updated Grammar)
4. Update ThiranInterpreter.py (to execute the constructed AST using the symbol table)

CHAPTER 4 - THIRAN - SETUP & USAGE

4.1 MINIMUM REQUIREMENTS FOR THIRAN:

- Any 64-bit Windows - 7 or higher running computer.
- 50 MB of minimum disk space for the software.
- No additional software or program is required to run this standalone software.

Note: Thiran is designed to run only on the Windows platform and not on Linux or Mac OS.

4.2 INSTALLING THIRAN:

Thiran has its own standalone setup which can be downloaded from the official GitHub repository for Thiran:

https://github.com/Surya-NarayananS/Thiran-Programming-Language/tree/main/Thiran Setup/

Once downloaded, double-click Setup.exe and follow through with the installer, and install Thiran.

Finally, to verify if Thiran was installed successfully, open a command prompt terminal anywhere, type the word thiran and press enter. If the terminal shows "*திறன் நிரல் மொழி - தொகுத்து இயக்க ஏதேனும்* '.ti' file*யை இணைக்கவும் !*", the setup installed correctly.

If it doesn't show as above, try adding the installed folder path to the system path environment variable and try again.

4.3 RUNNING OUR FIRST PROGRAM:

- Now, create a new file with the ".ti" extension anywhere you want (example: myprogram.ti), and type the following code in the file by opening it with any text editor:

```
காட்டு("உலகுக்கு வணக்கம்!")
காட்டு("தமிழில் ப்ரோக்ராம்மிங் எளிது!")
```

- Save the file and open a command prompt terminal in the same folder where the ".ti" file is, and type the command given below and press enter to execute the file.

```
thiran yourfilename.ti
```

- The terminal outputs the following:

```
உலகுக்கு வணக்கம்!
தமிழில் ப்ரோக்ராம்மிங் எளிது!
```

- Congratulations, you successfully ran your first Thiran program.

CHAPTER 5 - THIRAN TUTORIAL

5.1 *எண்கள்* (NUMBERS):

In Thiran, *எண்கள்* (or numbers) are of two types:

- ***முழுஎண்***
- ***புள்ளிஎண்***

முழுஎண் (int):

முழுஎண் include all integer numbers. Examples of *முழுஎண்* are 3, 21, 0, -2, -7.

புள்ளிஎண் (float):

புள்ளிஎண் are numbers that have a fractional part. Examples of *புள்ளிஎண்* are 2.1, 3.0, -17.26, 0.0.

5.2 *வாக்கியம்* (STRING):

String (or *வாக்கியம்*) is any text surrounded by double quotes (" ").

Examples:

- "*இது ஒரு வாக்கியம்*"
- "Hello World! :)"
- "ABcd123"
- "Pi *இன் மதிப்பு* 3.14159"

Here, Thiran will not treat 3.14159 as a "*புள்ளிஎண்*" but rather as a "*வாக்கியம்*" due to its enclosure within double-quotes (" ").

Using double quotes (") inside a string (*வாக்கியம்*):

Using double quotes (") inside a string may accidentally terminate the string (*வாக்கியம்*). So, to display (") double quotes without terminating the string, we can use '\' (backslash) before the double quotes (").

Example:

- "I may go to "Chennai" tomorrow." → will result in an error as the string gets split into:
    - "I may go to " → a string
    - Chennai → not a string
    - " tomorrow." → a string

To overcome this, we can use '\' before double quotes (") as follows:

- "I may go to \"Chennai\" tomorrow." → will result in a single string: "I may go to "Chennai" tomorrow."

Inserting a new line inside a string (*வாக்கியம்*):

We can insert a new line inside a string by using "\n" as follows:

- "Hello\nWorld\n!"

The above will result into:

Hello

World

!

Inserting a tab space inside a string (*வாக்கியம்*):

We can insert a tab space inside a string by using "\t" as follows:

- "Hello\tWorld\t!"

The above will result into:

```
Hello World !
```

5.3 OPERATORS:

In Thiran, we use operators to perform various operations. The different types of operators available in Thiran are:

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operator

5.3.1 ARITHMETIC OPERATORS:

Arithmetic operators perform mathematical operations. The various arithmetic operators in Thiran are:

- '+' (for addition / கூட்டல்)
- '-' (for subtraction / கழித்தல்)
- '*' (for multiplication / பெருக்கல்)
- '/' (for division / வகுத்தல்)
- '%' (for modulo (remainder of a division) / வகுத்தல் மீதம்)

Examples using Arithmetic Operators:

- 2 + 3 → 5
- 4 - 7 → -3
- 2.3 * 0.5 → 1.15
- -3 / 1.5 → -2.0
- 13 % 2 → 1
- 6 % 3 → 0

In an arithmetic operation that includes both number types (*முழுஎண்* and *புள்ளிஎண்*), *முழுஎண்* is automatically converted to *புள்ளிஎண்*. During *வகுத்தல்* '/', the *முழுஎண்* is automatically converted to *புள்ளிஎண்*, even though both numbers may be of type *முழுஎண்*.

Examples:

- -3 / 1.5 gets converted into -3.0 / 1.5
- 2 + 3.2 gets converted into 2.0 + 3.2
- 1.5 * 0 gets converted into 1.5 * 0.0
- 2 / 1 gets converted into 2.0 / 1.0
- -17 / 2 gets converted into -17.0 / 2.0
- 13 % 2.0 gets converted into 13.0 % 2.0

Note: Division by zero (0) is not possible and an attempt to do so will show an error:

- 12 / 0 → *கணித பிழை: வகுக்கும் எண்* '0' *ஆக இருக்க முடியாது*!

Using '+' with *வாக்கியம்* (string):

The arithmetic operator '+' can be used for combining multiple strings (*வாக்கியங்கள்*) into a single *வாக்கியம்* (string).

Examples:

- "Hello" + " World" → "Hello World"
- "*வண*" + "*க்கம்*" → "*வணக்கம்*"
- "123" + "456" → "123456"

Here, "123" + "456" gives "123456" because both "123" and "456" are of type - *வாக்கியம்*. Hence a string containing numbers should not be treated as numbers.

Note: *வாக்கியம்* (string) doesn't support the other arithmetic operators -, *, /, %. An attempt to do so will result in an error:

- "ABC" - "abc123" → *பிழை*: *வாக்கியத்துடன் பொருந்தா* Binary operation
- "ABC" * "abc123" → *பிழை*: *வாக்கியத்துடன் பொருந்தா* Binary operation
- "ABC" / "abc123" → *பிழை*: *வாக்கியத்துடன் பொருந்தா* Binary operation

Chaining multiple operators:

In Thiran, multiple Arithmetic operators can be chained together.

Examples:

- 2 + 3 * 4.5 / 2 - 2
- 3 * 2 + 1 % 5
- 3.2 * 2 + (3 * 4 / 2)

Thiran, while executing the above examples, will follow operator precedence and associativity. Parenthesis - ( ) gives the operators inside them higher precedence.

Operator Precedence:

Operator precedence decides which arithmetic operation should be performed first.

The precedence of Operators is given below:

1. () (Parenthesis) - Highest Precedence (Execution: left to right)

Example:

- 1 * 2 + (2 - 1) → 3

Here, the contents inside the parenthesis '()' have the highest precedence followed by '*' and then by '+'.

So, for 1 * 2 + (2 - 1), the execution goes like this:

- 1 * 2 + (2 - 1) → 1 * 2 + 1
- 1 * 2 + 1 → 2 + 1
- 2 + 1 → 3
- 3

2. '/' (வகுத்தல்), '*' (பெருக்கல்) and '%' (வகுத்தல் மீதம்) - Same Precedence (Execution: left to right)

Example:

- 3 * 2 / 2 * 2 % 2 → 0.0

Here, '*' and '/' have the same precedence. So, the execution goes from left to right.

For 3 * 2 / 2 * 2, the execution goes like this:

- 3 * 2 / 2 * 2 % 2 → 6 / 2 * 2 % 2
- 6 / 2 * 2 % 2 → 3.0 * 2 % 2
- 3.0 * 2 % 2 → 6.0 % 2
- 6.0 % 2 → 0.0
- 0.0

3. '+' (கூட்டல்) and '-' (கழித்தல்) - Same Precedence (Execution: left to right)

Example:

- 3 + 2 - 2 + 2 → 5

Here, '+' and '-' have the same precedence. So, the execution goes from left to right.

For 3 + 2 - 2 + 2, the execution goes like this:

- 3 + 2 - 2 + 2 → 5 - 2 + 2
- 5 - 2 + 2 → 3 + 2
- 3 + 2 → 5
- 5

An example that includes multiple arithmetic operators:

- 2 * (3 - (12 - -5)) + (2 * (5 + 3)) → -12

The execution for the above goes like this:

- 2 * (3 - (12 - -5)) + (2 * (5 + 3))
  - (3 - (12 - -5)) → 3 - 17
  - 3 - 17 → -14
- 2 * -14 + (2 * (5 + 3))
  - (2 * (5 + 3)) → 2 * 8
  - 2 * 8 → 16
- 2 * -14 + 16 → -28 + 16
- -28 + 16 → -12
- -12

5.3.2 COMPARISON OPERATORS:

Comparison operators perform a comparison between two entities. It gives out one of these two values based on the outcome of the comparison:

- *சரி* (True)
- *தவறு* (False)

The entities can be:

- Numbers (example: 12, -3, 5, 0)
- Arithmetic operations that evaluate to a number (example: 3+2, 4-4, 3*2)
- Variables

The various comparison operators in Thiran are:

- == (Equality)
- != (Inequality)
- < (Less than)
- > (Greater than)
- <= (Less than or equal to)
- >= (Greater than or equal to)

== (Equality):

== is used to check if the value on the left-hand side is equal to the value on the right-hand side.

If the values are:

- Equal: it returns *சரி*
- Not equal: it returns *தவறு*

Examples:

- 2 == 2 → *சரி*
- 3 == 2 → *தவறு*
- -1 == 1 → *தவறு*
- 0 == 0 → *சரி*

!= (Inequality):

!= is used to check if the value on the left-hand side is not equal to the value on the right-hand side.

If the values are:

- Not Equal: it returns *சரி*
- Equal: it returns *தவறு*

Examples:

- 2 != 2 → *தவறு*
- 3 != 2 → *சரி*
- -1 != 1 → *சரி*
- 0 != 0 → *தவறு*

< (Less than):

< is used to check if the value on the left-hand side is less than the value on the right-hand side.

If the value on the left is:

- Less than the value on the right: it returns *சரி*
- Greater or equal to the value on the right: it returns *தவறு*

Examples:

- 2 < 2 → *தவறு*
- 3 < 2 → *தவறு*
- -1 < 1 → *சரி*
- 17 < 20 → *சரி*

> (Greater than):

> is used to check if the value on the left-hand side is greater than the value on the right-hand side.

If the value on the left is:

- Greater than the value on the right: it returns *சரி*
- Less or equal to the value on the right: it returns *தவறு*

Examples:

- 2 > 2 → *தவறு*
- 3 > 2 → *சரி*
- -1 > 1 → *தவறு*
- 20 > 17 → *சரி*

<= (Less than or equal to):

<= is used to check if the value on the left-hand side is less than or equal to the value on the right-hand side.

If the value on the left is:

- Less than or equal to the value on the right: it returns *சரி*
- Greater than the value on the right: it returns *தவறு*

Examples:

- 2 <= 2 → *சரி*
- 3 <= 2 → *தவறு*
- -1 <= 1 → *சரி*
- 20 <= 17 → *தவறு*

>= (Greater than or equal to):

>= is used to check if the value on the left-hand side is greater than or equal to the value on the right-hand side.

If the value on the left is:

- Greater than or equal to the value on the right: it returns *சரி*
- Less than the value on the right: it returns *தவறு*

Examples:

- 2 >= 2 → *சரி*
- 3 >= 2 → *சரி*
- -1 >= 1 → *தவறு*
- 17 >= 20 → *தவறு*

5.3.3 LOGICAL OPERATORS:

Logical operators can combine two or more comparison statements. Logical operators evaluate to either *சரி* (True) or *தவறு* (False). Logical operators can be used to build logic in the program. In Thiran, there are three logical operators:

- *மற்றும்* (and)
- *அல்லது* (or)
- *இல்லை* (not)

***மற்றும்* (and):**

மற்றும் returns *சரி* when the conditions to the left and right of '*மற்றும்*' evaluate to *சரி*, otherwise, it returns *தவறு*.

The four possible combinations of *மற்றும்* are:

- *சரி மற்றும் சரி → சரி*
- *சரி மற்றும் தவறு → தவறு*
- *தவறு மற்றும் சரி → தவறு*
- *தவறு மற்றும் தவறு → தவறு*

Examples:

- 2!=3 *மற்றும்* -1==-1 → *சரி*
- 0==1 *மற்றும்* 0<=4 → *தவறு*
- 7<=7 *மற்றும்* -3>-6 → *சரி*
- 3==3 *மற்றும்* 7>5 → *தவறு*

***அல்லது* (or):**

அல்லது returns *சரி* when at least one of the conditions to the left and right of '*அல்லது* ' evaluate to *சரி*, otherwise, it returns *தவறு*.

The four possible combinations of *அல்லது* are:

- *சரி அல்லது சரி → சரி*
- *சரி அல்லது தவறு → சரி*
- *தவறு அல்லது சரி → சரி*
- *தவறு அல்லது தவறு → தவறு*

Examples:

- 2<3 *அல்லது* -1==-1 → *சரி*
- 0!=1 *அல்லது* 0>=4 → *சரி*
- 7<=7 *அல்லது* -3>-6 → *சரி*
- 1 != 1 *அல்லது* 5>7 → *தவறு*

இல்லை (not):

இல்லை flips *சரி* into *தவறு* and *தவறு* into *சரி* respectively.

The two possible combinations of இல்லை are:

- இல்லை *சரி* → *தவறு*
- இல்லை *தவறு* → *சரி*

Examples:

- இல்லை -1==-1 → *தவறு*
- இல்லை 0>=4 → *சரி*
- இல்லை 7<=7 → *தவறு*
- இல்லை 1 != 1 → *சரி*

Chaining Logical Operations:

Multiple Logical Operations can be chained to create complex logic.

Examples:

- 3<2 *அல்லது* இல்லை 3>2
- 7<2 *அல்லது* இல்லை 5>2 *மற்றும்* 1==1
- இல்லை 1>-1 *மற்றும்* 10==10

Thiran, while executing the above examples, will follow logical operator precedence. Parenthesis '( )' gives the logical operators inside them higher precedence.

Logical Operator Precedence:

Logical Operator precedence decides which logical operation should be performed first.

The precedence of Logical Operators is given below:

1. () (Parenthesis) - 1st Precedence (Execution: left to right):

Example:

- இல்லை (2==2 அல்லது 3==2) → தவறு

Here, the contents inside the parenthesis '()' have the highest precedence.

For இல்லை (2==2 அல்லது 3==2), the execution goes like this:

- இல்லை (2==2 அல்லது 3==2) → இல்லை (சரி)
- இல்லை (சரி) → தவறு
- தவறு

2. இல்லை (not) - 2nd Precedence (Execution: left to right):

Example:

- இல்லை 2<=1 மற்றும் 3==3 → சரி

Here, "இல்லை" has the highest precedence.

So, for இல்லை 2<=1 மற்றும் 3==3, the execution goes like this:

- இல்லை 2<=1 மற்றும் 3==3 → சரி மற்றும் 3==3
- சரி மற்றும் 3==3 → சரி
- சரி

3. மற்றும் - 3rd Precedence (Execution: left to right):

Example:

- 3!=3 அல்லது 2==2 மற்றும் 2!=2 → தவறு

Here, "மற்றும்" has the highest precedence.

So, for 3!=3 அல்லது 2==2 மற்றும் 2!=2, the execution goes like this:

- 3!=3 அல்லது 2==2 மற்றும் 2!=2 → 3!=3 அல்லது தவறு
- 3!=3 அல்லது தவறு → தவறு
- தவறு

4. அல்லது - Last Precedence (Execution: left to right):

அல்லது has the least precedence of all the logical operators.

Note: If we want அல்லது to execute first, then we can do so by surrounding it with parenthesis - ( ).

An example that includes multiple logical operators:

- 7<2 அல்லது இல்லை 5>2 மற்றும் 1==1 → தவறு

The execution for the above goes like this:

- 7<2 அல்லது இல்லை 5>2 மற்றும் 1==1
    - இல்லை 5>2 → இல்லை சரி
    - இல்லை சரி → தவறு
    - தவறு
- 7<2 அல்லது தவறு மற்றும் 1==1
    - தவறு மற்றும் 1==1 → தவறு மற்றும் சரி
    - தவறு மற்றும் சரி → தவறு
    - தவறு
- 7<2 அல்லது தவறு → தவறு அல்லது தவறு
- தவறு அல்லது தவறு → தவறு
- தவறு

5.3.4 ASSIGNMENT OPERATOR:

The assignment operator assigns value to variables (Refer to next Section). In Thiran, '=' (Equal to) is the only assignment operator.

Examples:

- *நீளம்* = 12
- *எண்கள்* = [0, 2, 3.14, 7, -3]
- *பெயர்* = "*திறன்*"

5.4 USING VARIABLES IN THIRAN:

Variables are like containers that store data values. In Thiran, variables can be declared as follows:

- *நீளம்* = 12

Here, "*நீளம்*" is the variable name. It has the value 12 stored in it. Now, "*நீளம்*" can be called throughout the program using its name.

Once declared, a variable can be called throughout the program. Whenever a variable is called, it returns the value stored in it.

Example:

```
நீளம் = 12

அகலம் = 5

பரப்பளவு = நீளம் * அகலம்
```

In the above statement, the "*பரப்பளவு*" variable will store: (the value of "*நீளம்*" variable) * (the value of "*அகலம்*" variable) = 12 * 5 = 60

So, the "*பரப்பளவு*" variable will have 60 (12 * 5) stored in it.

A variable's value can be changed throughout the program.

Example:

```
a = 5 → (here, 'a' is declared with a value of 5)
b = 10.2
a = b → (now 'a' has the value 10.2 (the value of 'b'))
a = "Thiran" → (now 'a' has the value "Thiran")
a = 3 → (now 'a' has the value 3)
```

Note: As shown in the above examples, variables should be declared before calling. If a variable is called without declaration, then it will result in an error.

Example:

```
a = 1
c = a + b
```

Here, the variable 'b' is not declared. So, it will result in an error: *அறிவிக்கப்படாத பெயர் பயன்பாடு*: b

In Thiran, the following rules should be followed while naming a variable:

- Variable names must start with a letter.
- Variable names can contain letters, numbers, and underscore (_).
- Variable names cannot start with a number or '_' (underscore)
- Variable names can contain both Tamil and English letters.
- Variable names cannot contain whitespaces ( ) or special characters other than underscore (_).
- Keywords cannot be used as variable names.

For example:

The below variable names are valid:

- myvar = 1
- my_var =1
- myvar1 = 1
- myvar_1 = 1
- myVar = 1
- MYVARIABLE = 1
- MyVar = 1

The below variable names are invalid and will result in an error:

- 1myvar= 1
- _myvar = 1
- my-var = 1
- my var = 1
- *அல்லது* = 1
- *தவறு* = 1

Here, *அல்லது* and *தவறு* are keywords reserved for special purposes. So, they cannot be used as variable names.

5.5 KEYWORDS IN THIRAN:

Keywords are special words, reserved for specific purposes in Thiran. They cannot be used as variable names. The function of each of these keywords is demonstrated in the rest of the documentation.

The keywords in Thiran are:

- மற்றும்
- அல்லது
- இல்லை
- சரி
- தவறு
- சரியெனில்
- அல்லதெனில்
- தவறெனில்
- முடி
- திரும்பச்செய்
- உண்மைவரை
- செயல்பாகம்
- வெளிகொடு
- தொடர்
- நிறுத்து
- இணை
- காட்டு
- வாங்கு

5.6 INPUT / OUTPUT IN THIRAN:

5.6.1 OUTPUT - காட்டு():

In Thiran, we can output values using the காட்டு() function.

Example:

```
கமட்ஞ(10) → will output the argument 10 to the console
கமட்ஞ("Hello") → will output the argument Hello to the console
```

The கமட்ஞ() function, after outputting its arguments will go to the next line.

Example:

```
கமட்ஞ("All")
கமட்ஞ("Is")
கமட்ஞ("Well")
```

The output for the above code will be:

All

Is

Well

கமட்ஞ() function can take multiple arguments. Each argument should be separated by a comma (,).

- கமட்ஞ(arg1, arg2, arg3, arg4, ...)

Each argument is evaluated and the result is printed to the console.

The different types of arguments in கமட்ஞ( ) are:

1. கமட்ஞ(variable) → outputs the value stored in the variable:

Examples:

```
a = 10
b = "Good Day"
c = [1, 2, 3]
```

காட்டு(a) → will output 10 to the console

காட்டு(b) → will output Good Day to the console

காட்டு(c) → will output [1, 2, 3] to the console

2. காட்டு(string) → outputs the string:

Example:

காட்டு("வணக்கம்") → will output வணக்கம் to the console

3. காட்டு(முழுஎண்) → outputs the number:

Example:

காட்டு(-12) → will output -12 to the console

4. காட்டு(புள்ளிஎண்) → outputs the number:

Example:

காட்டு(0.7) → will output 0.7 to the console

4. காட்டு(expression) → outputs the result after evaluating the expression:

Example:

காட்டு(2 + 3 - 1) → will output 4 to the console

காட்டு(2 < 3) → will output சரி to the console

காட்டு(இல்லை தவறு மற்றும் சரி) → will output சரி to the console

Examples of காட்டு() with multiple arguments:

காட்டு("hello", 2, [1, 2, 3], -7)

Output: hello 2 [1, 2, 3] -7

```
name = "Surya"
காட்டு("Hi", name + "!")
```

Output:

Hi Surya!

Here, the 2nd argument to *காட்டு*() uses the '+' operator to join two strings. (Refer to the section - "Using '+' with *வாக்கியம்* (string)")

```
காட்டு(2 + 3, இல்லை தவறு, 2 > 0, "string1", "string2")
```

Output:

5 *சரி சரி* string1 string2

```
காட்டு("New \nLine", "\t", 2.0)
```

Output:

New

Line 2.0

5.6.2 INPUT - *வாங்கு*():

In Thiran, we can take input from the console using the *வாங்கு*() function.

We usually assign the *வாங்கு*() to a variable to store the inputted value in the variable.

வாங்கு( ) optionally takes a single argument, which it will output to the console before getting input.

Examples:

```
num = வாங்கு("Enter a number: ")
காட்டு("The number you entered is:", num)
```

Note: The input value is stored in the variable "num" which can be used throughout the program.

Output:

Enter a number: 12.5 → user enters 12.5

The number you entered is: 12.5

```
காட்டு("Enter your Grade:")
grade = வாங்கு()
காட்டு("The grade is:", grade)
```

Output:

Enter your Grade:

A → user enters A

The grade is: A

5.7 CONTROL FLOW STATEMENTS:

Control flow statements alter the flow of the program. These statements generally evaluate a condition and based on the result (*சரி* or *தவறு*) alter the flow of the program. In Thiran, we have three types of control flow statements:

- Decision-Making Statements
- Looping Statements
- Jump Statements

5.7.1 DECISION-MAKING STATEMENTS:

The decision-making statements, execute a block of code based on a condition. In Thiran, the following three keywords are used for decision-making statements:

- ***சரியெனில்* (if)**
- ***அல்லதெனில்* (else if)**
- ***தவறெனில்* (else)**

The four possible decision-making combinations are:

- *சரியெனில்*()
- *சரியெனில்*() *தவறெனில்*
- *சரியெனில்*() *அல்லதெனில்*() ... *தவறெனில்*
- *சரியெனில்*() (*உள்ளே சரியெனில்*() ...)

1. *சரியெனில்* ():

The *சரியெனில்*() executes a block of code if the condition inside the parenthesis evaluates to *சரி*, otherwise, it skips the block of code altogether.

A *சரியெனில்*() block should always end with the keyword "*முடி*".

Syntax of *சரியெனில்*():

```
சரியெனில்(condition):
    ... (will execute only if the condition is true)
முடி
```

Examples:

```
எண் = 10
சரியெனில்(எண் % 2 == 0):
```

```
    காட்டு(எண், "is an even number")
முடி
```

Output: 10 is an even number

```
எண் = 7
சரியெனில்(எண் % 2 == 0):
    காட்டு(எண், "is an even number")
முடி
```

Output: (Nothing will be outputted as the *சரியெனில்*() evaluated to *தவறு*)

2. *சரியெனில்() தவறெனில்*:

In *சரியெனில்*() *தவறெனில்*, if the condition inside the parenthesis evaluates to:

- *சரி*, then the block of code below *சரியெனில்* gets executed; the block of code below *தவறெனில்* is skipped altogether.
- *தவறு*, then the block of code below *தவறெனில்* gets executed; the block of code below *சரியெனில்* is skipped altogether.

A *சரியெனில்*() *தவறெனில்* block should always end with the keyword "*முடி*".

Syntax of *சரியெனில்() தவறெனில்*:

```
சரியெனில்(condition):
    ... (will execute only if the condition is true)
தவறெனில்:
    ... (will execute only if the condition is false)
முடி
```

Examples:

```
எண் = 10
சரியெனில்(எண் % 2 == 0):
    காட்டு(எண், "is an even number")
தவறெனில்:
    காட்டு(எண், "is an odd number")
முடி
```

Output: 10 is an even number

```
எண் = 7
சரியெனில்(எண் % 2 == 0):
    காட்டு(எண், "is an even number")
தவறெனில்:
    காட்டு(எண், "is an odd number")
முடி
```

Output: 7 is an odd number

3. *சரியெனில்() அல்லதெனில்() ... தவறெனில்*:

In *சரியெனில்*() *அல்லதெனில்*() ... *தவறெனில்*, if the condition inside the parenthesis of *சரியெனில்* evaluates to:

- *சரி*, then the block of code below *சரியெனில்* gets executed; the blocks of code below *அல்லதெனில்* and *தவறெனில்* are skipped altogether.
- *தவறு*, then the condition inside the parenthesis of *அல்லதெனில்*, that follows, gets evaluated:

- If it evaluates to *சரி*, it behaves similarly to *சரியெனில்* by only executing the block of code below it while skipping the other blocks.
- If it evaluates to *தவறு*, it skips its block and moves on to the next *அல்லதெனில்* (if available) and repeats the same. If there is no *அல்லதெனில்* available, then the block of code below *தவறெனில்* gets executed.

Note: According to the need, there can be many *அல்லதெனில்*() blocks between *சரியெனில்*() and *தவறெனில்* blocks.

A *சரியெனில்*() *அல்லதெனில்*() ... *தவறெனில்* block should always end with the keyword "*முடி*".

Syntax of *சரியெனில்*() *அல்லதெனில்*() ... *தவறெனில்*:

```
சரியெனில்(condition1):
    ... (will execute only if the condition1 is true)
அல்லதெனில்(condition2):
    ... (will execute only if the condition2 is true)
அல்லதெனில்(condition3):
    ... (will execute only if the condition3 is true)
:
தவறெனில்:
    ... (will execute only if all the conditions are false)
முடி
```

Examples:

```
எண் = 10
சரியெனில்(எண் > 0):
```

```
    காட்டு(எண், "is a positive number")
அல்லதெனில்(எண் < 0):
    காட்டு(எண், "is a negative number")
தவறெனில்:
    காட்டு(எண், "is neither positive nor negative")
முடி
```

Output: 10 is a positive number

```
எண் = -3
சரியெனில்(எண் > 0):
    காட்டு(எண், "is a positive number")
அல்லதெனில்(எண் < 0):
    காட்டு(எண், "is a negative number")
தவறெனில்:
    காட்டு(எண், "is neither positive nor negative")
முடி
```

Output: -3 is a negative number

```
எண் = 0
சரியெனில்(எண் > 0):
    காட்டு(எண், "is a positive number")
அல்லதெனில்(எண் < 0):
    காட்டு(எண், "is a negative number")
தவறெனில்:
    காட்டு(எண், "is neither positive nor negative")
```

```
முடி
```

Output: 0 is neither positive nor negative

4. *சரியெனில்*() (*உள்ளே சரியெனில்*() ...)

Inside *சரியெனில்*() block of code, there can be another decision-making statement like:

- *சரியெனில்*()
- *சரியெனில்*() *தவறெனில்*
- *சரியெனில்*() *அல்லதெனில்*() ... *தவறெனில்*

which, in turn, may contain another decision-making statement inside it.

If the condition inside the parenthesis of the first *சரியெனில்* evaluates to *சரி*, then the block of code below it gets executed. The decision-making statement inside the block of code will follow the same.

Note: According to the need, there can be many decision-making statements inside a *சரியெனில்*() block.

In *சரியெனில்*() (*உள்ளே சரியெனில்*() ...), each *சரியெனில்* block should end with its own "*முடி*" keyword.

Syntax of *சரியெனில்*() (*உள்ளே சரியெனில்*() ...):

```
சரியெனில்(condition1):
    ... (will execute only if the condition1 is true)
    சரியெனில்(condition2):
        ... ( will execute if the condition2 is true)
    முடி
    ...
முடி
```

Examples:

```
எண் = 8
சரியெனில்(எண் >= 0):
    காட்டு(எண், "is a positive number")
    சரியெனில்(எண் > 5):
        காட்டு(எண், "is above 5")
    முடி
தவறெனில்:
    காட்டு(எண், "is Negative")
முடி
```

Output:

8 is a positive number

8 is above 5

```
எண் = 10
சரியெனில்(எண் % 2 == 0):
    சரியெனில்(எண் > 0):
        காட்டு(எண், "is a positive even number")
    அல்லதெனில்(எண் < 0):
        காட்டு(எண், "is a negative even number")
    தவறெனில்:
        காட்டு(எண், "is Zero")
    முடி
தவறெனில்:
```

```
    காட்டு(எண், "is an odd number")
முடி
```

Output:

10 is a positive even number

5.7.2 LOOPING STATEMENTS:

Looping statements execute a block of code, some pre-determined times (or) continually until a condition evaluates to *தவறு*.

In Thiran, we have two kinds of looping statements:

- *திரும்பச்செய்* (for loop)
- *உண்மைவரை* (while loop)

1. *திரும்பச்செய்* (for loop):

The *திரும்பச்செய்* executes a block of code, 'n' number of times. The 'n' value is expected inside the parenthesis - ( ). The 'n' value can either be a positive *முழுஎண்* or 0. *புள்ளிஎண்* is not accepted and will result in an error. A *திரும்பச்செய்* block should always end with the keyword "*முடி*"

Note: If the 'n' value is 0 or a negative *முழுஎண்*, then the entire *திரும்பச்செய்* block will not be executed.

Syntax for *திரும்பச்செய்*:

```
திரும்பச்செய்(n):
    … (executes n times)
முடி
```

Examples:

```
திரும்பச்செய்(3):
    காட்டு("கணினி அறிவியல்!")
முடி
```

Output:

கணினி அறிவியல்!

கணினி அறிவியல்!

கணினி அறிவியல்!

```
திரும்பச்செய்(2):
    num = வாங்கு("Enter a number: ")
    காட்டு("You entered ", num)
முடி
```

Output:

Enter a number: 7

You entered 7

Enter a number: 0

You entered 0

```
திரும்பச்செய்(0):
    காட்டு("I don't run!")
முடி
```

Output:

(Nothing will be outputted)

2. *உண்மைவரை* (while loop):

The *உண்மைவரை* executes a block of code repeatedly until a condition evaluates to *தவறு*.

The condition to be evaluated is expected inside the parenthesis - ( ).

The condition inside the parenthesis is evaluated. If the condition evaluates to:

- *சரி*, then the block of code below it will be executed. (Note: If executing the block of code, at any point, make the condition change to *தவறு*, then execution is stopped immediately, and the loop exits). If the block of code executes completely without changing the condition to *தவறு*, then the condition is evaluated again, and the process repeats.
- *தவறு*, then the block of code below it won't be executed. (The loop exits)

We can also pass a number (*முழுஎண்* or *புள்ளிஎண்*) inside the parenthesis - (). Every possible number other than 0 and 0.0, will by default evaluate to *சரி*. So, passing a number inside ( ) should be discouraged, as it may lead to an infinite loop.

A *உண்மைவரை* block should always end with the keyword "*முடி*"

Warning: A condition (Example: 3<4, *சரி*, 1, etc.) that can never be *தவறு*, will make the *உண்மைவரை* to execute the block of code below it, infinitely. This is termed an **infinite loop**. Infinite loops should be **avoided at all costs unless specifically needed**.

During program execution, inside the command prompt, if you accidentally enter an infinite loop, pressing ctrl+c will terminate the program altogether.

Syntax for *உண்மைவரை*:

```
உண்மைவரை(condition):
    ... (executes until the condition is true)
முடி
```

Examples:

```
x = 1
உண்மைவரை(x <= 3):
    காட்டு("The value is: ", x)
    x = x + 1
முடி
```

Output:

The value is: 1

The value is: 2

The value is: 3

```
x = 1
உண்மைவரை(x):
    காட்டு("I run only once!")
    x = 0  →  (Here, we change the condition to be தவறு)
முடி
```

Output:

I run only once!

```
உண்மைவரை(0):
    காட்டு("I don't run!")
முடி
```

Output:

(Nothing will be outputted)

5.7.3 JUMP STATEMENTS

In Thiran, jump statements alter the flow of execution of loops. So, jump statements can be used only inside a loop (*உண்மைவரை* or *திரும்பச்செய்*). The two jump statements available are:

- *நிறுத்து* (break)
- *தொடர்* (continue)

Note: Using jump statements outside a loop will result in an error.

1. *நிறுத்து* (break):

The "*நிறுத்து*" keyword can only be used inside a loop (*உண்மைவரை* or *திரும்பச்செய்*). The *நிறுத்து* keyword, when executed, ends the loop immediately.

Note: If a loop contains another loop inside it and *நிறுத்து* has been used in the innermost loop, it will end only the innermost loop and the outermost loop will continue as usual.

Examples:

```
x = 0
திரும்பச்செய்(10):
    x = x + 1
```

```
    சரியெனில்(x == 5):
        நிறுத்து
    முடி
முடி
காட்டு("The value of x:", x)
```

Output:

The value of x: 5

```
உண்மைவரை(சரி):
    num = வாங்கு("Enter 10: ")
    சரியெனில்(num == 10):
        நிறுத்து
    முடி
முடி
```

Output:

Enter 10: -2

Enter 10: 3.2

Enter 10: 7

Enter 10: 10

```
திரும்பச்செய்(2):
    காட்டு("வெளியே")
    x = 0
    உண்மைவரை(சரி):
        காட்டு(" - உள்ளே")
```

```
        x = x + 1
        சரியெனில்(x == 3):
            நிறுத்து
        முடி
    முடி
முடி
```

Output:

```
வெளியே
 - உள்ளே
 - உள்ளே
 - உள்ளே
வெளியே
 - உள்ளே
 - உள்ளே
 - உள்ளே
```

2. தொடர் (continue):

The "தொடர்" keyword can only be used inside a loop (உண்மைவரை or திரும்பச்செய்). The தொடர் keyword, when executed, skips the remaining block of code below it and goes to the next iteration.

Examples:

```
x = 0
திரும்பச்செய்(5):
    x = x + 2
```

```
        சரியெனில்(x == 4):
            தொடர்
        முடி
        காட்டு(x)
முடி
```

Output:

2

6

8

10

```
x = 0
உண்மைவரை(x < 5):
    x = x + 1
    சரியெனில்(x == 2):
        காட்டு("I am 2")
        தொடர்
    முடி
    காட்டு("I am not 2")
முடி
```

Output:

I am not 2

I am 2

I am not 2

I am not 2

I am not 2

5.8 FUNCTIONS (செயல்பாகம்) IN THIRAN:

A function (செயல்பாகம்) is a block of code, that runs whenever it is called. When a block of code repeats multiple times, it is generally advised, to make that block of code a function and call it wherever needed.

Defining a function (செயல்பாகம்):

In Thiran, we can define a function using the keyword, "செயல்பாகம்".

The syntax for defining a function:

```
செயல்பாகம் function_name ():
    ... (code)
முடி
```

Note: Every function definition should end with the keyword "முடி".

Example:

```
செயல்பாகம் greet():
    காட்டு("வணக்கம்!")
முடி
```

Calling a defined function:

Once a function is defined, it can be called any number of times, using the function_name followed by parenthesis - ()

Example:

```
செயல்பாகம் greet():
    காட்டு("வணக்கம்!")
முடி
greet()
greet()
greet()
```

The "greet()" will execute the block of code defined under the "greet" function.

Output:

வணக்கம்!

வணக்கம்!

வணக்கம்!

Passing arguments to function:

We can pass some data into functions for processing. These data are called arguments.

While defining a function, the arguments needed are specified inside the parenthesis ( ). If there are multiple arguments, they can be separated by commas (,).

```
செயல்பாகம் function_name (argument1, argument2, ...):
    ... (code - can use argument1, argument2, ...)
முடி
```

Once an argument is specified inside the parenthesis ( ), it can be used inside the function definition by using its name.

Note: Arguments act just like variables inside a function definition.

Example:

```
செயல்பாகம் greet(name):
    காட்டு("வணக்கம்", name)
முடி
```

```
செயல்பாகம் add(num1, num2):
    காட்டு(num1 + num2)
முடி
```

Once a function has been defined with arguments, then, the values for those arguments must be passed while calling that function.

Also, the number of values passed while calling a function should match the number of arguments defined in the function.

Failing to meet the above two, will result in an error.

Examples:

```
செயல்பாகம் greet(name):
    காட்டு("வணக்கம்", name)
முடி
greet("Surya")
greet() → (Error) No value passed
greet("Plato", "Aristotle") → (Error) 2 values passed while 1 is expected
```

Output:

வணக்கம் Surya

(Error)

(Error)

```
செயல்பாகம் add(num1, num2):
காட்டு(num1 + num2)
முடி
add(1, 2)
add (2, 3)
add(1) → (Error) 1 value is passed while 2 are expected
add() → (Error) No value passed
add(1, 2, 3) → (Error) 3 values are passed while 2 are expected
```

Output:

3

5

(Error)

(Error)

(Error)

Using "வெளிகொடு" inside functions:

A function can return a value back to the function call. To return a value from a function, the keyword "வெளிகொடு" is used, followed by the return value.

Syntax:

```
செயல்பாகம் function_name():
    ... (code)
    வெளிகொடு value
முடி
```

Note: The keyword "வெளிகொடு" can only be used inside a function definition.

Examples:

```
செயல்பாகம் add(num1, num2):
    வெளிகொடு num1 + num2
முடி
result = add(1, 3)
காட்டு(result)
```

Output:

4

Here, add(1, 3) will return 4, which is then stored in the variable result.

```
செயல்பாகம் mul(num1, num2):
    வெளிகொடு num1 * num2
முடி
result = mul(1, 3) * 5
காட்டு(result)
```

Output:

15

Here, mul(1, 3) will return 3, which is then multiplied by 5 and then stored in the variable result.

A thing to note about function definitions:

Things declared inside a function definition work only within that function definition. In other words, if a variable is declared inside a function definition, calling that variable outside the function will throw an error.

Example:

```
செயல்பாகம் add(num1, num2):
    sum = num1 + num2
    வெளிகொடு sum
முடி
result = add(1, 2)
காட்டு(sum) → will result in an error
```

In Thiran, for simplicity, the following restrictions have been put forth:

1. Inside a function call, passing another function call as an argument isn't allowed. (The exceptions to this rule are: *காட்டு*() and *வாங்கு*() functions)

Example:

```
func1(func2()) → will result in an error
```

2. We cannot pass a function call statement as a condition inside decision-making statements.

Example:

```
சரியெனில்(func()): → will result in an error
    ... (code)
முடி
```

Built-in Functions:

Built-in functions are functions that have already been defined for you to use readily. In Thiran, we have many useful built-in functions. They can be called directly anywhere in the program.

The various built-in functions in Thiran are:

- வெளியேறு()
- வகை()
- முழுஎண்()
- புள்ளிஎண்()
- வாக்கியம்()
- round()
- power()
- பட்டியல்_இணை()
- பட்டியல்_நீக்கு()
- பட்டியல்_சேர்()
- பட்டியல்_சுருக்கு()
- பட்டியல்_நீளம்()
- get_pi()
- get_e()
- get_tau()
- ceil()
- floor()
- sin()
- cos()
- tan()
- factorial()
- mod()
- gcd()
- lcm()
- sqrt()
- radians()
- degrees()
- randint()
- choice()
- random()

Examples:

```
a = 2
காட்டு(வகை(a))
```

Output:

```
முழுஎண்
```

Here, வகை() is a built-in function that returns its argument's type.

```
a = முழுஎண்(21.3)
காட்டு(a)
```

Output:

```
21
```

Here, முழுஎண்() is a built-in function that converts its argument into a முழுஎண் and returns it.

Note: In Thiran, we can overwrite a built-in function. To do so, just define your function with the built-in function's name, and the next time, you call that function, your custom function gets executed instead of the built-in function.

Example to overwrite built-in வகை() function:

```
செயல்பாகம் வகை():
    காட்டு("Overwritten வகை()")
முடி
வகை()
```

Output:

```
Overwritten வகை()
```

Recursion:

In Thiran, a function can call itself. If a function calls itself, the function is called a recursive function.

Example:

```
செயல்பாகம் factorial(n):
    சரியெனில்(n==1):
        வெளிகொடு 1
    முடி
    வெளிகொடு n * factorial(n-1) → Here, the function calls itself.
முடி
காட்டு(factorial(5))
```

Output:

120

5.9 பட்டியல் (LIST):

A list (*பட்டியல்*), as the name suggests, is a list of elements or items stored in a variable. The elements or items can be of different types. Lists are useful when there is a need to store multiple elements.

Creating and using lists:

A list can be created using square brackets - [ ]. The elements inside the square brackets should be separated using commas (,).

Example:

```
list1 = [1, -3.2, "Thiran", 0]
list2 = [5]
empty_list = []
கபட்ஜ(list1)
கபட்ஜ(list2)
கபட்ஜ(empty_list)
```

Output:

```
[1, -3.2, 'Thiran', 0]
[5]
[]
```

Note: A List can have duplicate elements or items.

Example:

```
list = [1, 2, 3, 2, 1, 2] →  allows duplicate elements
```

Accessing list elements:

In a list, each element has an index associated with it. The index starts from **0**.

- 1st element's index- 0
- 2nd element's index - 1
- 3rd element's index - 2
- ... (and so on)

Example:

```
list = ["a", "b", "c", "d"]
```

Here,

- Index of 1st element "a" = 0
- Index of 2nd element "b" = 1
- Index of 3rd element "c" = 2
- Index of 4th element "d" = 3

So, if a list has four elements, its elements' indexes are 0, 1, 2, and 3. Now, to access individual elements in a list, we can use its corresponding index. The syntax for list indexing is given below:

Syntax:

```
List_Name[index] (where the index can be 0 or above)
```

Note: List index other than a positive (+ve) முழுஎண் will result in an error.

Example:

```
list1 = [1, -3.2, "Thiran", 0]
கமட்டு(list1[0])
கமட்டு(list1[1])
கமட்டு(list1[3])
கமட்டு(["Hello", "Hi"][1])
கமட்டு(list1[0] + list1[1])
கமட்டு(list1[0] * 5)
கமட்டு(list1[4]) →  (Here, index 4 doesn't exist in list1)
```

Output:

```
1
-3.2
```

0

Hi

-2.2

5

(Error)

Joining two lists:

In Thiran, we can join two lists using the '+' arithmetic operator.

Examples:

```
add = [1, 2, 3] + [4, 5, 6]
காட்டு(add)
```

Output:

[1, 2, 3, 4, 5, 6]

```
a = [1, 2]
b = ["a", "b"]
c = a + b
காட்டு(c)
```

Output:

[1, 2, 'a', 'b']

Note: பட்டியல் (list) doesn't support the other arithmetic operators -, *, /, %. An attempt to do so will result in an error:

List manipulation using Built-in functions:

In Thiran, for list manipulation, we have five built-in functions:

- *பட்டியல்_நீளம்*()
- *பட்டியல்_இணை*()
- *பட்டியல்_சேர்*()
- *பட்டியல்_நீக்கு*()
- *பட்டியல்_சுருக்கு*()

***பட்டியல்_நீளம்*():**

பட்டியல்_நீளம்() is a built-in function that takes a list (*பட்டியல்*) as an argument and returns the list's length (the number of elements in the list).

Example:

```
myList = [2.3, 3, -1, "Hi", 0, 2] → 6 elements
length = பட்டியல்_நீளம்(myList)
காட்டு(length)
```

Output:

6

```
காட்டு(பட்டியல்_நீளம்([1, 2]))
```

Output:

2

```
காட்டு(பட்டியல்_நீளம்([]))
```

Output:

0

Note: If a list is empty (no elements), its length is 0.

***பட்டியல்_இணை*():**

பட்டியல்_இணை() is a built-in function that takes in two arguments:

- a list (*பட்டியல்*)
- an element to add to the list

It adds the element at the end of the list. The function returns "None" because it adds the element directly to the list variable.

Example:

```
list1= [1, 2, 3]
list2 = ["a", "b"]
பட்டியல்_இணை(list1, 4)
பட்டியல்_இணை(list2, "c")
காட்டு(list1, list2)
```

Output:

[1, 2, 3, 4] ['a', 'b', 'c']

***பட்டியல்_சேர்*():**

பட்டியல்_சேர்() is a built-in function that takes in three arguments:

- a list (*பட்டியல்*)
- index at which the element should be added
- element to add to the list

It adds the element at the index specified. The function returns "None" because it adds the element directly to the list variable just like *பட்டியல்_இணை*().

Examples:

```
list1= [1, 2, 3]
list2 = ["a", "b"]
பட்டியல்_சேர்(list1, 0, 4)
பட்டியல்_சேர்(list2, 1, "c")
காட்டு(list1, list2)
```

Output:

```
[4, 1, 2, 3] ['a', 'c', 'b']
```

Note: The difference between *பட்டியல்_இணை*() and *பட்டியல்_சேர்*() is that *பட்டியல்_இணை* adds the element at the end but *பட்டியல்_சேர்* adds the element at the specified index.

***பட்டியல்_நீக்கு*():**

பட்டியல்_நீக்கு() is a built-in function that takes in two arguments:

- a list (*பட்டியல்*)
- element to remove from the list

It removes the **first occurrence** of the element in the list. If the element to remove is not in the list, an error is thrown.

The function returns "None" because it adds the element directly to the list variable.

Example:

```
list1= [1, 2, 3, 2, 1]
பட்டியல்_நீக்கு(list1, 2)
காட்டு(list1)
```

Output:

[1, 3, 2, 1]

Here, the first occurrence of element 2 in list1 is removed.

```
list1= [1, 2, 3, 2, 1]
பட்டியல்_நீக்கு(list1, 5) → 5 is not in list1
```

Output:

(Error)

பட்டியல்_சுருக்கு():

பட்டியல்_நீக்கு() is a built-in function that takes in a list (*பட்டியல்*) as an argument. It removes the last element in that list and returns that removed element.

Example:

```
list1= [1, 2, 3, 4, 5]
last_element = பட்டியல்_சுருக்கு(list1)
காட்டு(list1)
காட்டு(last_element)
```

Output:

[1, 2, 3, 4]

5

5.10 COMMENTS IN THIRAN:

Programmers use comments to explain how a code works, thereby increasing code readability. In Thiran, any text followed by '#' (hash) until the next line is considered a comment. **Thiran Interpreter ignores all comments** in the code.

Example:

```
# This is a comment
கமட்டு("Hello World!") # This line prints Hello World!
```

Output:

"Hello World!

```
# கமட்டு("Hello") →  code is written as a comment
```

Output:

(No output)

5.11 இணை (IMPORT) IN THIRAN:

In Thiran, we can import code from other ".ti" files. By importing a ".ti" file into our main Thiran file, we can directly use the functions and variables defined in that imported file. In this way, we can organize Thiran code into separate ".ti" files based on their functionality or purpose.

To import a ".ti" file into another file, the "இணை" keyword is used, followed by a list of file_names to be imported separated by commas (,). The Thiran file to be imported should be in the same directory as the Thiran file importing it.

Syntax:

இணை [filename1, filename2, ...]

Note: In இணை[ ] statement, filenames should not contain the ".ti" extension, i.e., if a file name is "maths.ti", we should only use "maths" by ignoring the ".ti" extension.

If Thiran couldn't find the file, an error is shown: "filename: *கண்டுபிடிக்க முடியவில்லை (ஒரே* path*ல் உள்ளதை உறுதிசெய்க*)"

Example:

If there are two Thiran files - main.ti and math.ti in the same folder or directory, we can import the math.ti into main.ti by:

(Code inside main.ti):

```
இணை[math]
```

Calling functions and variables from imported files:

In Thiran, importing a file will internally execute the entire file and store its definitions in the main file. So, if a file to be imported contains any input or output statement, it gets executed while importing.

Once a file is imported, its functions or variables can be called or used just like any normal function or variable.

Examples:

(Code inside file.ti):

```
செயல்பாகம் example():
    காட்டு("Imported Function!")
முடி
pi = 3.14
```

(Code inside main.ti):

```
இணை [file]
example() # calling imported function
காட்டு(pi) # calling imported variable
```

Output (executing just main.ti):

```
Imported Function!
3.14
```

Here, both the main.ti and the file.ti are inside the same directory or folder.

(Code inside maths.ti):

```
செயல்பாகம் pi():
    வெளிகொடு 3.14
முடி
காட்டு("I run while importing maths.ti")
```

(Code inside main.ti):

```
இணை [maths]
ans = pi() # calling imported function
காட்டு(ans)
```

Output (executing just main.ti):

```
I run while importing maths.ti
3.14
```

Here, both the main.ti and the maths.ti are inside the same directory or folder.

Import order:

In Thiran, modules are imported one after the other. So, if two modules have functions with the same name, calling the function from the main file will cause the latest imported module's function to be executed.

Example:

```
(Code inside file1.ti):
செயல்பாகம் print():
    காட்டு("File1's function")
முடி

(Code inside file2.ti):
செயல்பாகம் print():
    காட்டு("File2's function")
முடி

(Code inside main.ti):
இணை [file1, file2]
print() # calling imported function
```

Output (executing just main.ti):

```
File2's function
```

Here, main.ti, file1.ti, and file2.ti are all inside the same directory or folder.

CHAPTER 6 - EXAMPLE PROGRAMS

1. Input Output

```
பெயர் = வாங்கு("உங்கள் பெயரை உள்ளிடுக: ")
காட்டு("வணக்கம்", பெயர் + "!")
```

Output:

```
C:\Users\surya\Desktop>thiran InputOutput.ti
உங்கள் பெயரை உள்ளிடுக: Surya
வணக்கம் Surya!
```

2. Area of a rectangle

```
நீளம் = வாங்கு("செவ்வகத்தின் நீளம் உள்ளிடுக: ")
அகலம் = வாங்கு("செவ்வகத்தின் அகலம் உள்ளிடுக: ")

சரியெனில்(நீளம் < 0 அல்லது அகலம் < 0):
   காட்டு("தவறான மதிப்பு! ")
தவறெனில்:
   பரப்பளவு = நீளம் * அகலம்    # Area = l * b
   காட்டு("செவ்வகத்தின் பரப்பளவு:", பரப்பளவு, "அலகுகள்")
முடி
```

Output:

```
C:\Users\surya\Desktop>thiran Area_Rectangle.ti
செவ்வகத்தின் நீளம் உள்ளிடுக: 12
செவ்வகத்தின் அகலம் உள்ளிடுக: 2.5
செவ்வகத்தின் பரப்பளவு: 30.0 அலகுகள்
```

3. Basic calculator

```
கூட்டல் = "+"

கழித்தல் = "-"

பெருக்கல் = "*"

வகுத்தல் = "/"


உண்மைவரை(சரி):

    எண்1 = வாங்கு("முதல் எண் உள்ளிடுக: ")

    எண்2 = வாங்கு("இரண்டாம் எண் உள்ளிடுக: ")

    செயல் = வாங்கு("கணித செயல்முறையை உள்ளிடுக (+, -, *, /): ")


    சரியெனில்(செயல் == கூட்டல்):

        விடை = எண்1 + எண்2

    அல்லதெனில்(செயல் == கழித்தல்):

        விடை = எண்1 - எண்2

    அல்லதெனில் (செயல் == பெருக்கல்):

        விடை = எண்1 * எண்2

    அல்லதெனில்(செயல் == வகுத்தல்):

        சரியெனில்(எண்2 == 0 அல்லது எண்2 == 0.0):
```

```
        காட்டு("வகுக்கும் எண் '0' ஆக இருக்க முடியாது!")
        நிறுத்து
    முடி
    விடை = எண்1 / எண்2
தவறெனில்:
    காட்டு("தவறான உள்ளீடு!")
    நிறுத்து
முடி
காட்டு(எண்1, செயல், எண்2, "=", விடை)
முடி
```

Output:

```
C:\Users\surya\Desktop>thiran Basic_Calculator.ti
முதல் எண் உள்ளிடுக: 1
இரண்டாம் எண் உள்ளிடுக: 3.2
கணிதச் செயல்முறையை உள்ளிடுக (+, -, *, /): +
1 + 3.2 = 4.2
முதல் எண் உள்ளிடுக: -1
இரண்டாம் எண் உள்ளிடுக: 0
கணிதச் செயல்முறையை உள்ளிடுக (+, -, *, /): *
-1 * 0 = 0
முதல் எண் உள்ளிடுக:
இரண்டாம் எண் உள்ளிடுக:
கணிதச் செயல்முறையை உள்ளிடுக (+, -, *, /):
தவறான உள்ளீடு!
```

4. Factorial of a number

```
எண் = வாங்கு("எண் உள்ளிடுக: ")


உள்ளிட்ட_வகை = வகை(எண்)
```

```
சரியெனில்(உள்ளிட்ட_வகை != "முழுஎண்"):
    காட்டு("உள்ளிட்ட எண் 'முழுஎண்' ஆக இருக்க வேண்டும்")
        வெளியேறு()
முடி

சரியெனில்(எண் == 0):
    காட்டு(எண், "வின் factorial: 1")
அல்லதெனில்(எண் > 0):
    முடிவு = 1
    ஐ = 1
    திரும்பச்செய்(எண்):
        முடிவு = முடிவு * ஐ
        ஐ = ஐ + 1
    முடி
    காட்டு(எண், "இன் factorial:", முடிவு)
தவறெனில்:
    காட்டு("எண் +ve ஆக இருக்க வேண்டும்!")
முடி
```

Output:

```
C:\Users\surya\Desktop>thiran Factorial.ti
எண் உள்ளிடுக: 5
5 இன் factorial: 120
```

5. Check if a number is prime

```
எண் = வாங்கு("எண் உள்ளிடுக: ")
பகாஎண் = சரி

வகை = வகை(எண்)
சரியெனில்(வகை != "முழுஎண்"):
   காட்டு("உள்ளிட்ட எண் 'முழுஎண்' ஆக இருக்க வேண்டும்")
   வெளியேறு()
முடி

சரியெனில்(எண் == 1):
   பகாஎண் = தவறு
அல்லதெனில்(எண் > 1):
   ஐ = 2
   உண்மைவரை(ஐ < எண்):
      சரியெனில்(எண்%ஐ == 0):
         பகாஎண் = தவறு
         நிறுத்து
      முடி
      ஐ = ஐ + 1
   முடி
தவறெனில்:
   பகாஎண் = தவறு
முடி
```

```
சரியெனில்(பகாஎண்):
    காட்டு(எண், "ஒரு பகா எண்!")
தவறெனில்:
    காட்டு(எண், "பகா எண் அல்ல!")
முடி
```

Output:

```
C:\Users\surya\Desktop>thiran Prime_number.ti
எண் உள்ளிடுக: 12
12 பகா எண் அல்ல!

C:\Users\surya\Desktop>thiran Prime_number.ti
எண் உள்ளிடுக: 7
7 ஒரு பகா எண்!
```

6. Fibonacci Sequence

```
எண்ணிக்கை = வாங்கு("எத்தனை பிபனாச்சி எண்கள் வேண்டும்: ")
டி1 = 0
டி2 = 1

உள்ளிட்ட_வகை = வகை(எண்ணிக்கை)
சரியெனில்(உள்ளிட்ட_வகை != "முழுஎண்"):
    காட்டு("உள்ளிட்ட எண் 'முழுஎண்' ஆக இருக்க வேண்டும்")
    வெளியேறு()
முடி
```

```
சரியெனில்(எண்ணிக்கை <= 0):
    காட்டு("எண் +ve ஆக இருக்க வேண்டும்!")
அல்லதெனில்(எண்ணிக்கை == 1):
    காட்டு("பிபனாச்சி வரிசை:", டி1)
தவறெனில்:
    காட்டு("பிபனாச்சி வரிசை:")
    திரும்பச்செய்(எண்ணிக்கை):
        காட்டு(டி1)
        எண் = டி1 + டி2
        டி1 = டி2
        டி2 = எண்
    முடி
முடி
```

Output:

```
C:\Users\surya\Desktop>thiran Fibonacci_Sequence.ti
எத்தனை பிபனாச்சி எண்கள் வேண்டும்: 7
பிபனாச்சி வரிசை:
0
1
1
2
3
5
8
```

7. Reversing a number

```
எண் = வாங்கு("எண் உள்ளிடுக: ")
தலைகீழ்எண் = 0

வகை = வகை(எண்)
சரியெனில்(வகை != "முழுஎண்"):
   காட்டு("உள்ளிட்ட எண் 'முழுஎண்' ஆக இருக்க வேண்டும்")
   வெளியேறு()
முடி

உண்மைவரை (எண் != 0):
   தலைகீழ்எண் = தலைகீழ்எண் * 10 + எண்%10
   எண் = முழுஎண்(எண்/10)
முடி

காட்டு("தலைகீழ் எண்:", முழுஎண்(தலைகீழ்எண்))
```

Output:

```
C:\Users\surya\Desktop>thiran Reverse_a_num.ti
எண் உள்ளிடுக: 837241091
தலைகீழ் எண்: 190142738
```

8. Sum of n natural numbers

```
செயல்பாகம் கூட்டுத்தொகை(எண்):
        சரியெனில்(வகை(எண்) != "முழுஎண்" அல்லது எண் < 0):
```

```
        காட்டு("sum( )ல் முழுஎண் [0 அல்லது அதற்கு மேல்] எதிர்பார்க்கப்பட்டது !")
        வெளியேறு()
    முடி
    வெளிகொடு முழுஎண்(எண்*(எண்+1)/2) # n*(n+1)/2
முடி


காட்டு(கூட்டுத்தொகை(10))
காட்டு(கூட்டுத்தொகை(0))
காட்டு(கூட்டுத்தொகை(27))
காட்டு(கூட்டுத்தொகை(75))
```

Output:

```
C:\Users\surya\Desktop>thiran "Sum of n natural nos.ti"
55
0
378
2850
```

9. Linear Search

```
செயல்பாகம் தேடு(வரிசை, எண்):
    x = 0
    திரும்பச்செய்(நீளம்):
        சரியெனில்(வரிசை[x] == எண்):
            வெளிகொடு x
        முடி
        x = x + 1
```

```
        முடி
        வெளிகொடு -1
    முடி

    வரிசை = []
    நீளம் = வாங்கு("வரிசையின் நீளம் உள்ளிடுக: ")
    i = 1
    திரும்பச்செய்(நீளம்):
        சொற்றொடர் = வாக்கியம்(i) + " ஆம் எண் உள்ளிடுக: "
        எண் = வாங்கு(சொற்றொடர்)
        பட்டியல்_இணை(வரிசை, எண்)
        i = i + 1
    முடி

    தேடல்_எண் = வாங்கு("எந்த எண்னை தேட வேண்டும்: ")
    விளைவு = தேடு(வரிசை, தேடல்_எண்)

    சரியெனில்(விளைவு != -1):
        காட்டு("வரிசையில்", விளைவு+1, "ஆம் இடத்தில் உள்ளது:",
தேடல்_எண்)
    தவறெனில்:
        காட்டு("வரிசையில் இல்லை:", தேடல்_எண்)
    முடி
```

Output:

```
C:\Users\surya\Desktop>thiran "Linear Search.ti"
வரிசையின் நீளம் உள்ளிடுக: 5
1 ஆம் எண் உள்ளிடுக: 2
2 ஆம் எண் உள்ளிடுக: 8
3 ஆம் எண் உள்ளிடுக: -1
4 ஆம் எண் உள்ளிடுக: 0
5 ஆம் எண் உள்ளிடுக: 11.5
எந்த எண்னை தேட வேண்டும்: -1
வரிசையில் 3 ஆம் இடத்தில் உள்ளது: -1
```

10. Bubble Sort

```
வரிசை = []
நீளம் = வாங்கு("வரிசையின் நீளம் உள்ளிடுக: ")
i = 1

திரும்பச்செய்(நீளம்):
    சொற்றொடர் = வாக்கியம்(i) + " ஆம் எண் உள்ளிடுக: "
    எண் = வாங்கு(சொற்றொடர்)
    பட்டியல்_இணை(வரிசை, எண்)
    i = i + 1
முடி

காட்டு("வரிசை வகைபடுத்தும் முன்:", வரிசை)
திரும்பச்செய்(நீளம்-1):
    x = 0
    திரும்பச்செய்(நீளம்-1):
```

```
            சரியெனில்(வரிசை[x] > வரிசை[x+1]):
                பட்டியல்_சேர்(வரிசை, x+2, வரிசை[x])
                பட்டியல்_நீக்கு(வரிசை, வரிசை[x])
            முடி
            x = x + 1
        முடி
    முடி


    காட்டு("வரிசை வகைபடுத்திய பின்:",வரிசை)
```

Output:

```
C:\Users\surya\Desktop>thiran BubbleSort.ti
வரிசையின் நீளம் உள்ளிடுக: 6
1 ஆம் எண் உள்ளிடுக: 0
2 ஆம் எண் உள்ளிடுக: -3
3 ஆம் எண் உள்ளிடுக: 2
4 ஆம் எண் உள்ளிடுக: -6
5 ஆம் எண் உள்ளிடுக: 7
6 ஆம் எண் உள்ளிடுக: 1
வரிசை வகைபடுத்தும் முன்: [0, -3, 2, -6, 7, 1]
வரிசை வகைபடுத்திய பின்: [-3, 0, -6, 2, 1, 7]
```

CHAPTER 7 - HURDLES FACED WHILE DEVELOPING THIRAN PROGRAMMING LANGUAGE

Though developing Thiran was exciting, fun-filled, and satisfying, it did pose some challenges. The main hurdles we faced while developing Thiran are:

Compound statements' structure:

The sentence structure in English is Subject-Verb-Object (SVO). This SVO structure is reflected in compound statements (like if, while, etc.) of nearly every English programming language. On the other hand, the sentence structure in Tamil is Subject-Object-Verb (SOV). This SOV structure makes it complex to construct compound statements in Tamil.

Choosing Keywords:

In Thiran, the keywords were chosen based on simplicity. So, they need not translate exactly to their English counterparts.

For example:

- Python - print
- Ezhil - பதிப்பி
- Thiran - காட்டு (Here, காட்டு (lit. show), implies that something is shown in the output).

CHAPTER 8 - FUTURE WORK AND CONCLUSION

8.1 FUTURE WORK:

Thiran, as an open-source programming language, has tremendous scope in vernacular Computer Science education. With proper support from the Government, Thiran can be taught to middle school Tamil medium students. Also, the language can be extended further by adding support for the following:

- Exception Handling
- Bitwise Operators
- Switch Statement
- Exception Handling
- Bitwise Operators
- Switch Statement

8.2 CONCLUSION:

We like to conclude this project by quoting our National Education Policy, 2020:

"The fundamental principles that will guide both the education system at large, as well as the individual institutions within it are: [...] ***promoting multilingualism and the power of language in teaching and learning****"*

- National Education Policy, 2020

CHAPTER 9 - REFERENCES

1. *Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986. ISBN 0-201-10088-6*
2. *Dasgupta, Sayamindu, and Benjamin Mako Hill. "Learning to code in localized programming languages." Proceedings of the fourth (2017) ACM conference on learning@ scale. 2017.*
3. *Annamalai, Muthiah. "Ezhil: A Tamil Programming Language." arXiv preprint arXiv:0907.4960 (2009).*
4. *Raj, Adalbert Gerald Soosai, et al. "What Do Students Feel about Learning Programming Using Both English and Their Native Language?." 2017 International Conference on Learning and Teaching in Computing and Engineering (LaTICE). IEEE, 2017.*
5. *National Education Policy, 2020*
6. Ezhil - https://en.wikipedia.org/wiki/Ezhil_(programming_language)
7. Python documentation (Reference) - https://docs.python.org/3/
8. Let's Build a simple interpreter (Reference) - https://ruslanspivak.com/lsbasi-part1/
9. Ezhil Language Foundation (Reference) - https://github.com/Ezhil-Language-Foundation/Ezhil-Lang

ANNEXURE

GRAMMAR BEHIND THIRAN PROGRAMMING LANGUAGE:

The grammar, the Thiran Parser uses to parse the code can be viewed in the official GitHub repository for Thiran:

https://github.com/Surya-NarayananS/Thiran-Programming-Language

CODE BEHIND THIRAN PROGRAMMING LANGUAGE:

Thiran was coded completely in Python. The entire code can be viewed in the official GitHub repository for Thiran:

https://github.com/Surya-NarayananS/Thiran-Programming-Language