Technische
Hochschule
Rosenheim

Department of Computer Science

Master Computer Science

NerdDeck — A comparison of functional programming paradigms between Go and F#

Course work in *Concepts of programming langugages*

Winter term 2023/2024

from

Maximilian Gobbel

Declaration of Originality

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Rosenheim, January 11, 2024

Maximilian Gobbel

Abstract

This paper presents a comparative analysis of functional programming paradigms in Go and F#, focusing on key concepts such as Algebraic Data Types and First-class Functions. The *NerdDeck* flashcard application serves as a practical example to illustrate the application of functional programming principles in both languages. Despite inherent challenges arising from the impurity of both languages, the comparison reveals distinctions in their approaches, emphasizing F#'s native support for functional programming and Go's adaptation within its multi-paradigm framework.

The challenges encountered underscore the importance of balancing functional and imperative programming in each language, prompting a nuanced approach in selecting languages for projects prioritizing functional programming principles. Looking ahead, further exploration into writing purer functional code within projects like *NerdDeck* is proposed. This could involve delving deeper into functional programming concepts and considering the use of purely functional languages like Haskell. As programming languages and paradigms continue to evolve, ongoing advancements may present new opportunities and challenges for functional programming practitioners.

The insights gained from this comparative analysis contribute to a broader understanding of functional programming in diverse language ecosystems and provide a foundation for future exploration and research in this evolving field.

Keywords: Functional Programming, Go Programming Language, F# Programming Language, Software Engineering

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Purpose of the Project . . . 1
- 1.3 Why F#? . . . 2
- 1.4 Roadmap . . . 2

2 Project Overview **3**
- 2.1 Description of the *NerdDeck* flashcard Application . . . 3
- 2.2 Requirements . . . 3
- 2.3 Data . . . 3
- 2.4 Functionality Overview and User Interface . . . 4

3 Overview of functional programming **6**
- 3.1 Key principles . . . 6
- 3.2 Imperative vs. Declarative . . . 7

4 Introduction to Go and F# **8**
- 4.1 Overview of Go . . . 8
- 4.2 Overview of F# . . . 8

5 Comparison of functional concepts **9**
- 5.1 Algebraic Data Types . . . 9
- 5.2 First-class Functions . . . 10
- 5.3 Key Takeaway . . . 11

6 Conclusion **13**
- 6.1 Challenges . . . 13
- 6.2 Outlook . . . 13

A Appendix **14**

Bibliography **19**

1 Introduction

This paper is written for the master course *Concepts of Programming Languages* of the department of computer science in the Winter Term 23/24 at the Technical University of Applied Sciences Rosenheim.

1.1 Background

According to the November 2023 TIOBE Index data, there are currently over 150 programming languages in existence. The programming landscape includes a diverse array of languages, reflecting the dynamic and ever-evolving nature of the field.[Tio] When producing software, a software engineer must select a programming language that is suitable for a particular problem. This topic has been discussed for approximately 50 years.[Tha82] The reason for this is that each language has its own paradigms and concepts. While there is no perfect language that can solve all problems, one can choose a language that fits many problems or one that is perfect for a single problem. The goal of selecting the appropriate programming language is to find one that suits the requirements and context of a given problem. This paper compares Go and F# in the context of functional programming (FP) to cover this topic.

1.2 Purpose of the Project

The aim of this project is to analyze and compare the similarities and differences between two functional programming (FP) languages: Go[1] and F#[2]. Go was chosen as the initial language due to its consistent usage throughout the course, making it a mandatory inclusion in the project. In order to conduct a comparative analysis, it is essential to include a second programming language alongside Go. For this purpose, F# has been selected. The decision to include F# in this study was deliberate, as explained in section 1.3. By comparing Go and F#, this project aims to uncover the nuances and divergences in their approaches to FP paradigms, shedding light on their distinctive features, strengths, and potential use cases. Through this comparative exploration, the project aims to provide valuable insights into the FP landscape. It seeks to offer a nuanced understanding of the strengths and trade-offs associated with both languages.

1 Website: `https://go.dev` (Accessed on 12/02/2023)
2 Website: `https://dotnet.microsoft.com/languages/fsharp` (Accessed on 12/02/2023)

1.3 Why F#?

One of the main reasons for selecting F# is its status as a functional-first programming language. Unlike Go's multi-paradigm approach, F# is committed to functional principles, providing an excellent opportunity to explore the benefits of implementing code from a pure perspective. For a more comprehensive understanding of F#, including its syntax, features, and FP principles, please refer to section 4.2 and chapter 5.

1.4 Roadmap

Chapter 2 briefly discusses the *NerdDeck* flashcard Application (App) as a practical context for understanding FP in action. Chapter 3 provides an overview of FP principles, followed by an introduction to Go and F# of FP in chapter 4. Chapter 5 is the centerpiece of this paper, as it provides a focused exploration of two critical functional concepts: Algebraic Data Types and First-class Functions. This section uses code examples to illuminate nuances in implementation, providing a concise yet insightful comparative analysis. The paper concludes in chapter 6, summarizing challenges and outlook from both the project and the comparison. This streamlined roadmap ensures a comprehensive yet condensed examination of FP in Go and F#.

2 Project Overview

2.1 Description of the *NerdDeck* flashcard Application

This coding project aims to compare and contrast the use of FP in Go and F#. The *NerdDeck* App is designed to aid in memorization using flashcards. A similar product, Anki[1], already exists for this purpose, with most of its components written in Rust and Python. However, *NerdDeck* stands out as it is written twice, once in Go and once in F#, utilizing the FP paradigm. In section 4.2, F# will not pose any issues since it allows for functional-first code. However, since Go is an impure language, the objective is to write in a functional manner as much as possible. *NerdDeck* adheres to the project framework by using the Command Line Interface (CLI) as a Graphical User Interface (GUI), while Anki has its own developed interface.

2.2 Requirements

The App should enable a single user to utilize the SuperMemo 2.0 (SM2) algorithm, which is employed for a learning technique known as spaced repetition.[Sm2] It is worth noting that one of Anki's algorithms is also based on SM2.[2] The requirements for *NerdDeck*, being developed by a student, are written from a student's perspective and are listed in table 2.1. It is important to ensure that these requirements do not deviate significantly from those of other users of the App. The objective is to implement this twice, once in Go and then in F#, both in a functional style. The program should act as a Minimal Viable Product (MVP), allowing for a comparison of FP. To reduce complexity, only one deck exists where users can add new flashcards. Because of that and to reduce the complexity, only one deck exists where a user can add new flashcards. However, it is not yet possible to delete flashcards. This is a potential future implementation.

2.3 Data

For this project, a single JavaScript Object Notation (JSON) file is used as a simple and lightweight database to store flashcards. The JSON object in listing 2.1 serves as an example. While this approach is suitable for educational and illustrative purposes, it may not be appropriate for production usage due to limitations in scalability and concurrent access limitations. For a more robust database solution in a production environment,

1 Website: `https://apps.ankiweb.net` (Accessed on 12/03/2023)

2 Website: `https://faqs.ankiweb.net/what-spaced-repetition-algorithm.html` (Accessed on 12/03/2023)

Table 2.1 *NerdDeck* Requirements

ID	Requirement
1	As a student, I want to create a flashcard inside a deck.
2	As a student, I want to view my flashcards.
3	As a student, I would like to learn how to utilize the program.
4	As a student, I want each flashcard to have a unique combination of a question and an answer to avoid duplicates.
5	As a student, I would like to utilize a spaced repetition algorithm to enhance my learning with flashcards.

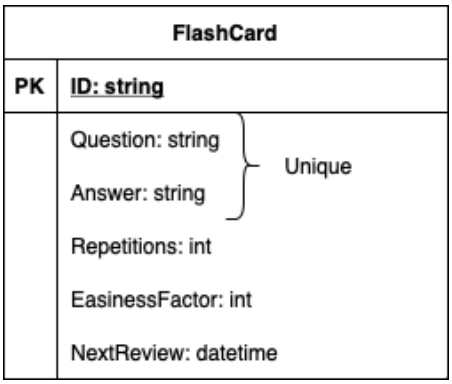

Figure 2.1 Diagram of the Flashcard model based on Unified Modeling Language (UML)

consider using a relational database (such as PostgreSQL[3] or MySQL[4]) or a NoSQL database (such as MongoDB[5]), depending on the specific requirements of the App.

Model

In order to meet the requirements outlined in table 2.1, a flashcard model is necessary. Figure 2.1 displays this model. It is crucial that users are able to create multiple flashcards and validate that there are no duplicates of the question-answer combination. The ID is generated from the question and answer of the flashcard, serving as a primary key for the model. This model should be used in both languages.

2.4 Functionality Overview and User Interface

Figure 2.2 displays the prompt inside the CLI of the F# implementation. The welcome screen and menu appear identical in the Go implementation. Users can choose from five options when executing the program:

3 Website: `https://www.postgresql.org` (Accessed on 12/02/2023)
4 Website: `https://www.mysql.com` (Accessed on 12/02/2023)
5 Website: `https://www.mongodb.com` (Accessed on 12/02/2023)

```json
[{
    "ID": "37268335dd6931045bdcdf926",
    "Question": "What algebraic data types does F# use?",
    "Answer": "Record types and discriminated unions",
    "Repetitions": 1,
    "EasinessFactor": 1.3,
    "NextReview": "2023-12-18T18:17:22.438077+01:00"
}]
```

Listing 2.1 Example of how a flashcard object is saved inside a JSON file

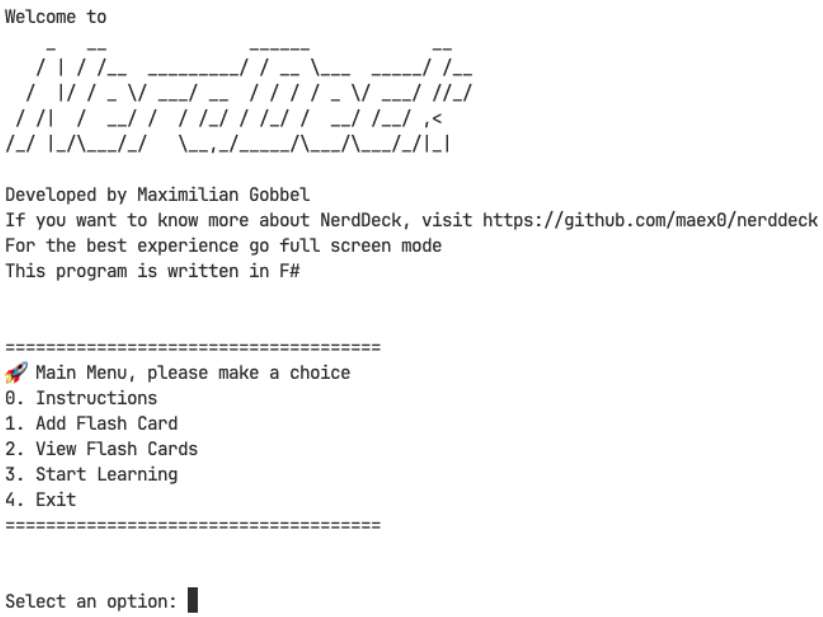

Figure 2.2 Screenshot of the CLI which shows the start of *NerdDeck*

- **0. Instructions:** Shows how the program should be used.
- **1. Add flashcard** Add a flashcard and saves it into the JSON file.
- **2. View flashcards:** Just prints all flashcards from the JSON file on the CLI.
- **3. Start Learning flashcards:** Loads all flashcards from the JSON file. Checks all due flashcards based on the next review. Learn every flashcard and then rate each one from 1 to 4 while 1 is bad and 4 is good. Apply the value as input for the SM2 algorithm and save the result in the JSON file.
- **4. Exit:** Exit the program.

3 Overview of functional programming

„As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write, easy to debug, and provides a collection of modules that can be re-used to reduce future programming costs. Conventional languages place conceptual limits on the way problems can be modularized. Functional languages push those limits back.“ [Hug89]

The motivation for using FP should be considered. Within the paradigm of FP, developers adhere to principles that prioritize the creation of expressive and concise solutions.

3.1 Key principles

Rooted in mathematical concepts like lambda calculus, FP emerges as a methodology characterized by rigor and elegance. Key principles integral to FP include:

- **Purity:** Strict avoidance of side effects to ensure deterministic behavior.
- **Higher-order functions:** Utilization of functions as parameters and return values for enhanced modularity.
- **Lazy evaluation:** Selective computation, evaluating values only when necessary for optimization.
- **Algebraic data types:** Incorporation of sum- and product-types, as discussed in more detail in Section 5.1.
- **First-class Functions:** A powerful technique, elaborated further in Section 5.2.

FP has a rich history dating back to 1950 with the introduction of the language LISP. Since then, FP has evolved across various programming languages, such as like Go and F#. F# strictly adheres to a functional-first approach, emphasizing immutability and mathematical principles. In contrast, Go incorporates impure functional elements, showcasing the adaptability of FP principles. Chapter 4 provides further insights into this topic. As developers navigate through contrasting algorithms, they reflect on the historical evolution and diversity of FP languages. This exploration serves as a testament to the flexibility of programming paradigms and prompts consideration of the nuanced trade-offs between explicit instruction and expressive abstraction within the dynamic context of FP. In conclusion, FP provides professionals with a robust set of tools, emphasizing

immutability, higher-order functions, and mathematical principles. FP not only addresses current challenges in software development but also promotes a deeper understanding of the complexities involved in designing robust, modular, and expressive code.

3.2 Imperative vs. Declarative

Within the programming paradigm spectrum, the imperative and declarative styles represent contrasting approaches to articulating code. This text will demonstrate both approaches using a simple algorithm written in pseudocode for summing up an array of numbers.

Imperative Approach

Algorithm 1 exemplifies the imperative paradigm. This paradigm directs the computer through computations using explicit steps. Each line of pseudocode provides instructions to the computer on how to perform the calculation, adhering to traditional imperative paradigms. The sum is explicitly calculated in a `for loop` from line three to five.

Algorithm 1: Imperative way of summing up an integer array

```
1 function SumArray (a);
  Input  : Integer Array a
  Output : Sum of elements in a
2 result ← 0
3 foreach element in a do
4 |   result ← result + element
5 end
6 return result
```

Declarative Approach

In contrast, the declarative style, demonstrated by Algorithm 2, emphasizes expressing the desired outcome directly rather than detailing step-by-step procedures. This approach is a hallmark of FP, leverages the power of functions and abstraction. In chapter 4, we will encounter a similar concise syntax while using F#.

Algorithm 2: Declarative way of summing up an integer array

```
1 function SumArray (a);
  Input  : Integer Array a
  Output : Sum of elements in a
2 return ∑_{element in a} element
```

4 Introduction to Go and F#

In this chapter, we explore the distinctive features of Go and F#, offering a brief overview of each language.

4.1 Overview of Go

It is widely used by large companies such Google, Paypal, and Netflix due to its ability to build simple, secure, and scalable systems. Go, also known as Golang, is a statically-typed and compiled programming language created by Google engineers Robert Griesemer, Rob Pike, and Ken Thompson in 2007. As an open-source language, Go is characterized by:

- **Concurrent programming:** Native support through *goroutines* and *channels* for scalable and parallelized Apps.
- **Efficiency:** Statically-typed, compiled that produces standalone binaries. Good for deployment in various environments.
- **Flexibility:** Can be used for Cloud & Network Services, CLIs, Web Development and Development Operations & Site Reliability Engineering.

[Gow]

4.2 Overview of F#

F# is a functional-first programming language developed by Microsoft Research in 2005. Alongside C# and Visual Basic, F# is another programming language within the .NET ecosystem. According to Microsoft's official website, it is an open-source language that enables the writing of succinct, robust, and performant code. Microsoft is as a leading contributor to the language. F# has an active community and support.[Fsh] While developing, F# has a powerful build-in feature called Read-eval-print loop (REPL). In addition, its key features include:

- **Functional-first:** Immutable by default, First-class functions, Pattern Matching, Algebraic Data Types.
- **Lightweight syntax:** Type inference and automatic generalization.
- **Interoperability:** Access .NET Libraries and APIs (e.g. ASP.NET or Entity Framework).

[Dot, Key]

5 Comparison of functional concepts

To compare the programming languages Go and F# in the context of FP, we will examine two concepts in more detail: Algebraic Data Types and First-class Functions.

5.1 Algebraic Data Types

Algebraic data types are fundamental in FP, as they allow for modeling complex data structures. However, in Go, these types are not directly supported, unlike in languages such as F#, which has native support for discriminated unions and pattern matching. In both Go and F#, the `FlashCard` type is utilized to represent a flashcard, capturing essential information such as the card's ID, question, answer, number of repetitions, ease factor, and the next review date. Although Go and F# serve the same purpose, there are syntactic and structural differences between their implementations.

F#

F# provides native support for algebraic data types through record types and discriminated unions. These allow developers to define a set of related values that can be handled concisely and expressively using pattern matching. This feature enhances the readability and maintainability of F# code, making it a powerful tool for developers who embrace FP.

Listing 5.2 defines the `FlashCard` from figure 2.1 using a record type[1]. The record contains fields for `ID`, `Question`, `Answer`, `Repetitions`, `EasinessFactor`, and `NextReview`. Records in F# are provided automatically, making them well-suited for representing data with fixed attributes.

Go

In Go, developers often use `struct` types to represent data structures. While they provide a way to encapsulate related data, they lack the succinct expressiveness of algebraic data types. Go's approach often involves using interfaces and composition to achieve similar outcomes, but this can lead to more verbose code compared to the concise syntax offered by algebraic data types.

Therefore, the `FlashCard` type is defined using a `struct`. This can be seen in listing 5.1. These types are mutable, which means that their fields can be updated after they are

1 Website: `https://learn.microsoft.com/dotnet/fsharp/language-reference/records` (Accessed on 01/08/2024)

created. Furthermore, Go employs explicit types for each field, resulting in a statically typed approach.

```go
type FlashCard struct {
    ID           string
    Question     string
    Answer       string
    Repetitions  int
    EasinessFactor      float64
    NextReview   time.Time
}
```

Listing (5.1) FlashCard representation in Go

```
type FlashCard =
    { ID : string
    Question : string
    Answer : string
    Repetitions : int
    EasinessFactor : float
    NextReview : DateTime }
```

Listing (5.2) FlashCard representation in F#

Figure 5.1 Comparison of FlashCard representations in Go and F#

5.2 First-class Functions

Both Go and F# support first-class functions, treating functions as first-class citizens. In Go, functions can be assigned to variables, passed as arguments, and returned as values. The use of anonymous functions, known as closures, is common in Go, enhancing the FP paradigm. Similarly, F# provides robust support for first-class functions, aligning with its FP paradigm. Functions in F# can be assigned to variables, passed as arguments, and returned as values. The language encourages the use of higher-order functions, enabling powerful abstractions through functions.

F#

Listing 5.3 provides an example in F# where a first-class function is defined named `isDueForReview` to check if a flashcard is due for review based on specific criteria. This function is then used to filter a list of flashcards to determine which ones are due for learning. The function should return a list of due flashcards. For instance, in line four, the function is called on the `List.filter` function. The function takes a predicate of type `T -> bool` and a list as input, and returns a list containing only the elements that satisfy the predicate.[2] Additionally, F# provides the pipe operator `|>` which allows developers to place the last argument at the forefront for convenience. This operator enables a smoother and more readable syntax, as demonstrate by the code `cards |> List.filter isDueForReview`, where the `isDueForReview` function is seamlessly applied to the `cards` list using the pipe operator. This expression is equivalent to line four of listing 5.3. The same functionality should now be implemented in the Go programming language.

2 Website: `https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html#filter` (Accessed on 01/08/2024)

```fsharp
let isDueForReview card =
    card.Repetitions > 0 && card.NextReview <= DateTime.Now

let dueFlashCards = List.filter isDueForReview cards
```

Listing 5.3 First-class function reresentation in F#

Go

Listing 5.1 provides a Go example that defines the first-class function `isDueForReview`, which is similar to the one above in the F# example. The `filterDueFlashCards` function on line six takes a slice of flashcards (`cards`) and applies the `isDueForReview` function to filter out only the flashcards that meet the review criteria. The filtered flashcards are then collected in the `dueFlashCards` slice and will be returned when all cards are covered.

For instance, on line 16, the function `filterDueFlashCards` is called with a given set of flashcards (`cards`) as input. The resulting list of flashcards due for review is then assigned to the variable `dueFlashCards`. This variable contains the outcome of the flashcard filtering process.

```go
func isDueForReview(card FlashCard) bool {
    return card.Repetitions > 0
    && card.NextReview.Before(time.Now())
}

func filterDueFlashCards(cards []FlashCard) []FlashCard {
    var dueFlashCards []FlashCard
    for _, card := range cards {
        if isDueForReview(card) {
            dueFlashCards = append(dueFlashCards, card)
        }
    }
    return dueFlashCards
}

dueFlashCards := filterDueFlashCards(cards)
```

Listing 5.4 First-class function reresentation in Go

5.3 Key Takeaway

Although Go's `struct` types are mutable by default, F#'s `record` types are immutable by default. Immutability simplifies reasoning about state and enables certain optimizations. It is worth noting that every field in Go must have an explicitly defined type, whereas in F#, they are inferred based on their usage. Neither language supports inheritance. Both languages support first-class functions, making them suitable choices for developers with

different preferences in balancing imperative and FP styles. The difference between the two languages lies in their syntax and expressiveness. This can be seen in each of the four examples in listing 5.1, 5.2, 5.3, and 5.4. It is worth noting that neither language is completely pure. While Go allows for mutation, even F#, which is primarily a functional language, permits its use.

6 Conclusion

This paper compares algebraic data types and first-class functions in the context of FP paradigms in Go and F#. The *NerdDeck* flashcard App is used as a practical example to illustrate the application of FP principles in both languages. The comparison reveals distinctions in their approaches, showcasing F#'s native support for FP and Go's adaptation within its multi-paradigm framework.

6.1 Challenges

The basis for a sound comparison in this paper is to write code that is as pure as possible to fulfill to the paradigm of FP. However, challenges arose due to the inherent impurity in both Go and F#. Although F# aligns more closely with FP principles, Go's multi-paradigm nature necessitated creative workarounds to achieve a functional style. This underscores the significance of balancing functional and imperative programming in each language. The challenges underscore the importance of a nuanced approach when selecting a language for projects that prioritize FP principles and their benefits.

6.2 Outlook

Looking ahead, there is potential for further exploration into writing more pure functional code within the *NerdDeck* or similar projects. To achieve this, a pure programming language such as Haskell[1] could be considered. This would involve delving deeper into FP concepts and finding innovative solutions within the constraints of a specific language. Furthermore, as programming languages and paradigms continue to evolve, ongoing advancements may present new opportunities and challenges for practitioners of FP. The insights gained from this comparative analysis contribute to a broader understanding of FP in diverse language ecosystems and provide a foundation for future exploration and research.

1 Website: `https://www.haskell.org` (Accessed on 12/20/2023)

A Appendix

Acronyms

App Application

CLI Command Line Interface

FP functional programming

GUI Graphical User Interface

JSON JavaScript Object Notation

MVP Minimal Viable Product

REPL Read-eval-print loop

SM2 SuperMemo 2.0

UML Unified Modeling Language

List of Figures

2.1 Diagram of the Flashcard model based on UML 4
2.2 Screenshot of the CLI which shows the start of *NerdDeck* 5

5.1 Comparison of FlashCard representations in Go and F# 10

Listings

2.1 Example of how a flashcard object is saved inside a JSON file 5

5.3 First-class function reresentation in F# 11
5.4 First-class function reresentation in Go 11

List of Tables

2.1 *NerdDeck* Requirements 4

Bibliography

[Dot] .NET programming languages | C#, F#, and Visual Basic. https://dotnet.microsoft.com/en-us/languages. (Accessed on 12/20/2023).

[Fsh] F# Software Foundation. https://fsharp.org/. (Accessed on 12/21/2023).

[Gow] The Go Programming Language. https://go.dev/. (Accessed on 11/27/2023).

[Hug89] J. Hughes. Why Functional Programming Matters. *The Computer Journal*, 32(2):98–107, feb 1989.

[Key] What is F# - .NET | Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/fsharp/what-is-fsharp. (Accessed on 12/21/2023).

[Sm2] Application of a computer to improve the results obtained in working with the SuperMemo method - SuperMemo. https://www.supermemo.com/en/blog/application-of-a-computer-to-improve-the-results-obtained-in-working-with-the-supermemo-method. (Accessed on 12/01/2023).

[Tha82] A. L. Tharp. Selecting the "Right" Programming Language. *SIGCSE Bull.*, 14(1):151–155, feb 1982.

[Tio] TIOBE Index - TIOBE. https://www.tiobe.com/tiobe-index/. (Accessed on 11/27/2023).