Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages
Assignment-01: Exploring Programming Paradigms

Roshni V

21st January, 2024

Understanding Paradigms

Understanding CS 60 Course - Paradigm Exploration

Introduction to CS 60 Course

The CS 60 course, taken during freshman year of college, provides an overview of computing topics. The course covers finite state automata, grammars, proveability, and complexity theory. It emphasizes learning four different programming languages representing different paradigms.

Key Takeaways from CS 60 Course

- The significance of abstraction in programming.
- Realizing the potential to become a polyglot programmer.
- Understanding the differences between programming languages and their suitability for different tasks.

Making Change Algorithm Overview

Introduction to the making change algorithm, focusing on a simpler example with denominations. Initial mention of the class and method structure for making change in various programming paradigms.

Transition to Functional Programming (FP) Paradigm

Overview of the functional programming (FP) paradigm, emphasizing pure functional programming. Introduction to syntax in Lisp family languages, specifically using Racket.

Functional Programming in Racket - Part 1

Explanation of prefix notation and its advantages. Introduction to defining functions in Racket using the ‘define‘ keyword. Demonstration of basic mathematical operations in functional programming.

Functional Programming in Racket - Part 2

Introduction to conditional statements (cond) in Racket. Demonstrating the use of cond for branching logic. Discussion of lists in Lisp languages, emphasizing the importance of parentheses.

Strengths of Functional Programming

Advantages of functional programming, especially in the context of concurrency. Ease of testing in functional programming due to the lack of state modification. Reusability of functional code across different programs.

Functional Programming Example: Making Change in Racket

Introduction to the functional programming example of making change in Racket. Breakdown of the function signature and its parameters. A brief explanation of the code structure for the making change algorithm in a functional paradigm.

Recursion in Ruby

The speaker explains a recursive solution in Ruby to make change using bills and coins. Conditions for the recursive calls include handling zero change, no money in the cash drawer, and dealing with different denominations.

Introduction to Prolog

An overview of Prolog, a logic and constraint programming language based on formal logic. Prolog programs consist of facts and clauses, describing "what" instead of "how" to solve a problem. Explanation of basic syntax, including variables, constants, facts, and rules.

Building Family Tree in Prolog

Demonstrates building a family tree in Prolog using facts and rules. Examples include defining fathers, mothers, and relationships between family members. Prolog's ability to answer queries about family relationships using pattern matching.

Recursive Rules in Prolog

Illustrates the creation of more complex rules in Prolog, specifically defining sibling relationships. Shows Prolog's strength in running programs both forwards and backwards. Highlights the power of pattern matching and logical implications in Prolog.

Lists and Pattern Matching in Prolog

Introduces Prolog's handling of lists using bar notation for pattern matching. Demonstrates a Prolog rule for checking membership in a list. The flexibility of Prolog's pattern matching, allowing for various list manipulations.

Making Change in Prolog

Presents a Prolog rule for making change, solving a problem similar to the recursive change-making in Ruby. Highlights Prolog's distinct approach, emphasizing constraints and logical implications. Discusses the absence of a return concept in Prolog and the necessity of setting up constraints for running programs backward.

Introduction to Assembly

Introduces assembly language, focusing on the limited vocabulary and registers (a and d). Explains the syntax, available computations, and the use of labels for constants. Emphasizes the absence of multiplication or division in the assembly language used in the discussion.

Assembly Language Basics

Assembly language essentials explained. Overview of registers, jumps, and basic computations. Limited vocabulary and operations in assembly programming.

Procedural Style in Assembly

Procedural style application in assembly programming. Explanation of loops, conditional jumps, and program flow. Limited branching logic and conditional statements in assembly.

Making Change in Assembly

Applying procedural style to make change in assembly. Using memory cells to represent the amount and coin denominations. Detailed walkthrough of the code for making change in assembly.

Paradigm 1: Logic Paradigm

The logic paradigm is centered around formal logic and mathematical reasoning. It involves defining relationships and rules to express a problem's solution logically. Key principles of the logic paradigm include:

0.1 Principles and Concepts

Research into the principles and concepts of the logic paradigm reveals a strong emphasis on declarative programming, where developers specify what needs to be accomplished without detailing how. Logic programming languages like Prolog exemplify this paradigm, relying on rules and facts to derive logical conclusions.

0.2 Case Study: Prolog

A case study on Prolog provides insight into the logic paradigm's application. Prolog excels in tasks that involve rule-based decision making, such as expert systems and natural language processing.

0.3 Logic Paradigm in ASP.NET

Integrating the logic paradigm into ASP.NET involves leveraging declarative constructs to handle application logic. Key considerations include:

0.4 Characteristics and Features

ASP.NET, primarily an imperative framework, incorporates logic paradigm principles through declarative constructs in markup languages like HTML and XML. This enables a separation of concerns and cleaner code organization.

0.5 Case Study: ASP.NET Web Forms

Analyzing ASP.NET Web Forms as a case study demonstrates how declarative controls and event-driven programming align with logic paradigm principles. The use of server controls allows expressing logic directly in the markup.

Language for Paradigm 1: ASP.NET

Understanding ASP.NET Core - Real World Project Exploration

Overview of ASP.NET Core and Course Introduction

In this introductory section, the course covers the development of a complete website with a database application using C. The focus is on CRUD (Create, Read, Update, Delete) operations in the context of a database. The instructor, Chad Sluter, introduces himself as a professor of computer science and software development.

Course Overview and Prerequisites

This section outlines the course topics, including ASP.NET Core, the Model-View-Controller (MVC) design pattern, database setup, registration, login, page security, and database querying. Prerequisites include an intermediate level of programming, prior experience with object-oriented programming, understanding of C, database setup, SQL queries, and knowledge of HTML, CSS, and JavaScript.

What is ASP.NET Core and Why Use It?

This section introduces ASP.NET Core, a framework developed by Microsoft for building web applications. A comparison with other frameworks (Java, PHP, Python, JavaScript) is provided, highlighting its popularity in enterprise-level applications. The evolution of ASP.NET from Active Server Pages (ASP) to ASP.NET Core is discussed, emphasizing its cross-platform and open-source nature.

Role of C in ASP.NET Core

C is introduced as the language used in ASP.NET Core development. The .NET framework, including its runtime engine and libraries, is explained. An overview of the full-stack environment in ASP.NET, combining back end, business logic, and front end, is provided.

ASP.NET Page Structure and Razor Markup

This section introduces ASP.NET pages and the Razor markup language. The end-to-end solution in ASP.NET, running both back end and front end, is explained. The flexibility of choosing front-end technologies like React, Angular, or Vue alongside ASP.NET is emphasized.

Building the Jokes App - Model, Views, and Controllers (MVC)

The Jokes App is introduced to demonstrate the Model-View-Controller (MVC) design pattern. Features such as adding, editing, and searching jokes are demonstrated, with an explanation of the MVC components: Model (class), Views (HTML pages), and Controllers (central logic).

Setting Up ASP.NET Core Project in Visual Studio

This section covers the process of setting up a new ASP.NET Core project in Visual Studio. It includes selecting project types, such as MVC for a full-stack solution, and configuring authentication options for individual user accounts.

Exploring the Generated ASP.NET Core Project

An overview of the solution explorer and SQL Server object explorer in Visual Studio is provided. Default files and folders generated by ASP.NET Core are explained. The application is launched, and default home and privacy pages are observed.

Modifying Views in ASP.NET Core

This section demonstrates the modification of HTML views (cshtml files) to customize content. Changes are made to the privacy and home pages, introducing the Razor syntax for combining C with HTML.

MVC in ASP.NET Core - Models, Views, and Controllers

The Model-View-Controller (MVC) design pattern is explained in detail. Components such as Model (class), Views (HTML pages), and Controllers (logic) are clarified. A visual representation of how MVC handles user requests and generates dynamic web pages is provided.

Creating Views and Models

The process of creating views in ASP.NET Core is demonstrated. Views, which are HTML code that gets data from the controller, are introduced. The Razor syntax for dynamic page creation is explained.

Creating a Model Class

The definition and purpose of a model in ASP.NET Core are discussed. A simple model class named "Joke" is created, illustrating properties and methods in a C class.

Generating Code with Controller

This section covers the generation of code using controllers in ASP.NET Core. Controllers for CRUD operations are created, and options while adding a new controller are overviewed.

Database Migrations

Enabling and using database migrations in ASP.NET Core are discussed. The purpose of migrations in creating database tables is explained, along with an introduction to the Package Manager Console and using commands.

ORM and Database Management

Object-Relational Mapping (ORM) in ASP.NET Core is explained. A comparison between ORM and traditional DAO (Data Access Objects) approaches is provided. The benefits and considerations of using an ORM are discussed.

Viewing Generated Database Tables

Inspecting and understanding the generated database tables is demonstrated. Checking the database structure in SQL Server Object Explorer and relating C classes (models) to database tables using ORM are covered.

Final Features and Overview

An overview of additional features added to the application is provided. Menu items for navigation are added, and the final product with improved functionalities is previewed.

Formulating Search Functionality

The issue with the "not found" error in the search functionality is discussed. The ‘show search form‘ method in the Jokes Controller is introduced, and initial steps in creating the form for searching jokes are demonstrated.

Troubleshooting View Not Found Error

The "view show search form was not found" error is explained. The creation of the search form view using Visual Studio is demonstrated, and running the application to test the created search form is covered.

Auto-generating a Search Form

Utilizing Visual Studio's "Add View" feature to generate a pre-programmed data input form is shown. Choosing the template for the form creation based on the desired functionality and adjusting the generated form are demonstrated.

Modifying Search Form UI

Modifying the auto-generated form to include only essential elements for the search functionality is demonstrated. Adjusting the input box, removing unnecessary code, and changing the submit button text are covered.

Creating Show Search Results

The ‘show search results‘ method in the Jokes Controller is introduced. Retrieving and handling the search phrase from the form for further processing is demonstrated. Initial testing using a simple string response to confirm data passing is covered.

Filtering Search Results

Transitioning from returning a simple string to a list of filtered search results is shown. Implementing a filter in the controller using the LINQ ‘where‘ clause and testing the search functionality with different search terms are covered.

Hiding Joke Answers in List View

Addressing the design decision to hide joke answers in the main jokes list view is explained. Utilizing Razor syntax to modify the presentation of the jokes in the index view and demonstrating changes in the UI are covered.

Adding Authorization for Create Action

Identifying the need for user authentication before allowing access to the create action is discussed. Implementing the ‘[Authorize]‘ attribute on the ‘create‘ method in the Jokes Controller and verifying authorization by testing the create action after modification are covered.

Implementing User Authentication

Introduction to user authentication and the login screen is provided. The registration process for creating an account and logging in to access features like creating new joke questions are demonstrated.

Securing Create Action with Authorization

Restricting access to the create page for unauthorized users is demonstrated. Implementing authorization for the ‘create‘ method in the Jokes Controller and explaining the significance of securing data entry forms with authorization are covered.

Adding Authorization to Data Processing Methods

Identifying the need for authorization in data processing methods like create, edit, and delete is discussed. Implementing the ‘[Authorize]‘ attribute on methods related to data processing and ensuring that security measures are in place for critical operations in the application are covered.

Introduction to CSS Styling

Exploration of the ‘wwwroot‘ folder and the importance of ‘site.css‘ for styling is covered. A simple CSS change, altering the background color of the body, is demonstrated. The distinction between modifying site-specific CSS and Bootstrap CSS is emphasized.

Basic CSS Styling Example

Making a simple background color change in the ‘site.css‘ file is demonstrated. The necessity of refreshing the browser to view CSS changes is explained. Users are encouraged to explore and customize styling through the ‘site.css‘ file.

Advanced Feature: User Column in Joke Questions

A brief introduction to the addition of a user column to associate authors with joke questions is provided. The advanced feature won’t be implemented in the current crash course. The achievements of the crash course, including custom navigation, table creation, search function, security integration, and CSS customization, are summarized.

Paradigm 2: Scripting Paradigm

The scripting paradigm focuses on automating tasks through scripting languages. It emphasizes quick development, ease of use, and adaptability. Key principles include:

0.6 Principles and Concepts

Scripting languages like Python and JavaScript are often associated with the scripting paradigm. These languages prioritize simplicity and readability, making them ideal for automation and rapid prototyping.

0.7 Case Study: Python

A case study on Python showcases the scripting paradigm’s effectiveness. Python’s syntax and extensive libraries contribute to its popularity in scripting scenarios, ranging from web development to data analysis.

0.8 Scripting Paradigm in PowerShell

PowerShell is a robust scripting language developed by Microsoft. Understanding its application within the scripting paradigm involves examining:

0.8.1 Characteristics and Features

PowerShell’s integration with the Windows environment and extensive commandlets make it a powerful scripting tool. The emphasis on object-oriented pipelines aligns with scripting paradigm principles.

0.8.2 Case Study: PowerShell Automation

Exploring how PowerShell is used for system administration tasks serves as a case study. Its scripting capabilities simplify automation, allowing administrators to perform complex operations with concise scripts.

Language for Paradigm 2: PowerShell

Understanding PowerShell - Real World Project Exploration

Introduction and Background

The session was conducted by Jim Tyler, a former Amazon engineer and K-12 Director of Technology. Jim provided a brief overview of his background and outlined the goal of teaching PowerShell within an hour.

1 PowerShell Basics

Jim commenced the tutorial with an introduction to PowerShell, defining it as both a command line shell and scripting language. Demonstrations included accessing PowerShell, adjusting font size, and checking the version.

2 PowerShell vs. Command Prompt

A comparison between PowerShell and Command Prompt functionalities was presented, showcasing basic commands like ping and cd in both environments. Jim highlighted the utility of PowerShell cmdlets and specialized functions.

3 Execution Policy in PowerShell

Jim explained the concept of execution policies and their impact on script execution. Demonstrations included setting and checking execution policies, ensuring proper configuration for running custom scripts.

4 Writing and Running PowerShell Scripts

Introduction to PowerShell Integrated Scripting Environment (ISE) was given, with a demonstration of creating and running a simple "Hello World" script. Emphasis was placed on the use of comments and tab completion.

5 Understanding Commandlets

Jim defined commandlets as predefined functions in PowerShell, following the verb-noun naming format. The exploration continued with the demonstration of different commandlets using "Get-Command."

6 Using Get-Help in PowerShell

The "Get-Help" command was introduced for accessing PowerShell documentation. Jim demonstrated how to get help for a specific commandlet (e.g., "Write-Host") and stressed the importance of understanding command parameters.

7 Piping Commands in PowerShell

Jim explained the concept of piping commands in PowerShell, showcasing the efficiency of piping output from one command into another. The power and flexibility of using pipelines in PowerShell were emphasized.

8 Working with PowerShell Variables

Variables in PowerShell, defined as containers for data, were introduced. Jim demonstrated creating and assigning values to variables using the dollar sign, with a basic example of invoking a variable.

9 PowerShell Variables and Types

PowerShell's support for different data types, including strings, integers, floating points, and booleans, was discussed. Demonstrations included setting and changing variable types using the "get type" method.

10 Everything in PowerShell is an Object

Jim highlighted PowerShell's adherence to the object-oriented paradigm, treating everything as an object. Variables, like the favorite character, were shown to have properties using the "get member" cmdlet.

11 Arrays in PowerShell

Arrays, defined as collections of variables grouped into a single variable, were introduced. Demonstrations covered creating, accessing, and manipulating arrays in PowerShell.

12 PowerShell Hash Tables

Hash tables, allowing the association of keys with values for organized data storage, were explored. Jim demonstrated the creation, modification, and removal of key-value pairs in hash tables.

13 Collecting User Input in PowerShell

The "Read-Host" cmdlet's introduction for collecting user input in PowerShell was provided. Examples showcased using "Read-Host" to gather information and set variables based on user input.

14 Conditional Statements with If-Then-Else

Jim explained and demonstrated conditional statements using the if-then-else structure in PowerShell. An example scenario involved evaluating the number of Pokémon caught to determine if someone is a Pokémon Master.

15 PowerShell Basics: If-Else Statements

Basic if-else statements in PowerShell were introduced, demonstrating a simple if statement, and showing the use of else and else-if statements to handle different conditions.

16 PowerShell Basics: Switch Statements

Jim introduced switch statements in PowerShell, showcasing their efficiency for evaluating data with multiple conditions. The cleaner syntax of switch statements compared to repetitive if-else blocks was emphasized.

17 PowerShell Basics: For Loops

The tutorial covered the introduction to for loops in PowerShell. The structure of a for loop and its three parameters were explained, with a demonstration of iterating through an array and printing each element.

18 PowerShell Basics: ForEach Loops

Jim introduced foreach loops in PowerShell, illustrating how they simplify iterating through elements in an array. A practical example demonstrated using foreach to process each element in an array.

19 PowerShell Basics: While Loops

The tutorial introduced while loops in PowerShell, explaining the structure and demonstrating their use for repeated execution based on a condition. A demonstration involved iterating through an array and printing each element.

20 PowerShell Basics: Do-While Loops

Jim explained do-while loops in PowerShell, describing the structure and demonstrating their use for repeated execution with condition checking at the end. A demonstration involved iterating through an array and printing each element.

21 Introduction to Defining Functions in PowerShell

The tutorial provided an overview of defining functions in PowerShell, with an example of creating a function named "test SpaceX" to ping SpaceX. Demonstrations included executing the defined function and emphasizing the importance of functions for repetitive tasks.

22 Creating Advanced Functions with Parameters

Jim introduced creating more advanced functions in PowerShell, demonstrating defining custom parameters inside the function (e.g., ping count). The use of commandlet bindings to make parameters mandatory and specify their data type was explained.

23 Error Handling and Exceptions in PowerShell

An overview of exceptions in PowerShell and scenarios where normal error handling is insufficient was given. The "throw" command for creating exceptions was introduced, with demonstrations of using "try-catch" blocks for effective error handling.

24 Working with Files and Directories in PowerShell

Basic file and directory operations in PowerShell were introduced, with demonstrations of commands like New-Item, Copy-Item, Move-Item, and Remove-Item for file and folder manipulation. The use of Test-Path to confirm the existence of a file or directory was explained.

25 Working with Active Directory in PowerShell

The tutorial covered working with Active Directory in PowerShell, demonstrating importing the Active Directory module. Examples showcased retrieving user information using Get-ADUser, setting variables, and modifying user attributes using Set-ADUser.

26 Changing User Attributes

Demonstrations included changing user attributes using PowerShell. The process of modifying the surname of the user "Frodo Baggins" to "Tyler" was illustrated. Emphasis was placed on understanding attribute names and exploring advanced features for a more comprehensive view.

27 Adding a User to a Group

The tutorial demonstrated adding a user to a group in Active Directory, showcasing the use of the Add-ADGroupMember commandlet to add "Frodo Baggins" to the "Fellowship" group. The simplicity of the command for managing group memberships was highlighted.

28 Advanced User Attributes and Adding a New User

The tutorial explored advanced user attributes in Active Directory using PowerShell. A walkthrough involved adding a new user, "Luke Skywalker," to the Active Directory with specified attributes. The process of setting a default password and enabling/disabling users was illustrated.

29 Resetting User Password

Demonstrations included how to reset a user password in Active Directory using PowerShell. The Set-ADAccountPassword commandlet was used to reset Luke Skywalker's password. The troubleshooting process when encountering issues was highlighted, emphasizing practical problem-solving in PowerShell.

Conclusion

The comprehensive tutorial by Jim Tyler provided an in-depth understanding of PowerShell, covering basics, scripting, and practical applications in managing systems and Active Directory. The code snippets and examples presented serve as valuable resources for users looking to enhance their PowerShell skills.

Comparison

30 Comparative Analysis

Comparing the logic paradigm with the scripting paradigm reveals distinct approaches to problem-solving. The logic paradigm prioritizes formal reasoning, making it suitable for rule-based systems. In contrast, the scripting paradigm focuses on practical automation, favoring simplicity and ease of use.

30.1 Strengths and Weaknesses

Each paradigm and associated language has its strengths and weaknesses. Logic paradigms excel in complex rule-based scenarios but may have a steeper learning curve. Scripting paradigms prioritize ease of use but might lack the formalism required for certain applications.

31 Challenges Faced

The exploration of programming paradigms presented challenges in terms of finding comprehensive case studies and extracting nuanced insights. Ensuring a balance between depth of research and effective presentation required careful consideration.

32 Conclusion

In conclusion, the logic paradigm and scripting paradigm offer distinctive approaches to programming, each with its merits. Integrating these paradigms into frameworks like ASP.NET and languages like PowerShell showcases their adaptability to various domains. A thoughtful comparative analysis provides valuable insights into their strengths, weaknesses, and application scenarios.

References

Include any references or sources you consulted for your assignment.