ICS C141: CONCEPTS OF PROGRAMMING LANGUAGES

Item	Value
Curriculum Committee Approval Date	11/17/2023
Top Code	070700 - Computer Software Development
Units	3 Total Units
Hours	72 Total Hours (Lecture Hours 54; Lab Hours 18)
Total Outside of Class Hours	0
Course Credit Status	Credit: Degree Applicable (D)
Material Fee	No
Basic Skills	Not Basic Skills (N)
Repeatable	No
Open Entry/Open Exit	No
Grading Policy	Standard Letter (S), • Pass/No Pass (B)

Course Description

This course delves into the theory and practical aspects of programming languages, exploring the fundamental concepts that underlie various programming paradigms. Students will gain a comprehensive understanding of the features, syntax, and semantics of programming languages and how they influence software development. Through a combination of theory, hands-on coding exercises, and analysis of programming languages, students will develop a deeper appreciation of language design and its impact on software engineering. ADVISORY: ICS C120. Transfer Credit: CSU; UC.

Course Level Student Learning Outcome(s)

1. Demonstrate the ability to differentiate between programming paradigms, such as imperative, functional, object-oriented, and logic programming.
2. Apply language design principles, including data abstraction and encapsulation.
3. Evaluate programming languages for specific tasks and understand trade-offs in language selection.

Course Objectives

- 1. Describe the historical development of programming languages and their role in computer science.
- 2. Give examples of programming languages for specific tasks and explain trade-offs in language selection.
- 3. Provide criteria for evaluating programming languages.
- 4. Explain the role of compilers and interpreters in transforming high-level code into executable programs.
- 5. Explain the principles of functional programming.
- 6. Explain the principles of object-oriented design.
- 7. Discuss the features, syntax, and semantics of programming languages and how they influence software development.

Lecture Content

Concepts of Programming Languages Reasons for Studying Concepts of Programming Languages Programming Domains Scientific Applications Business Applications Artificial Intelligence Web Software Programming Environments Evolution of the Major Programming Languages Zuse s Plankalk l Pseudocodes The IBM 704 and Fortran Functional Programming: Lisp Computerizing Business Records: COBOL Object-Oriented Programming Combining Imperative and Object-Oriented Features: C++ An Imperative-Based Object-Oriented Language: Java Scripting Languages The Flagship .NET Language: C# Markup-Programming Hybrid Languages Describing Syntax and Semantics The General Problem of Describing Syntax Formal Methods of Describing Syntax Describing the Meanings of Programs: Dynamic Semantics Lexical and Syntax Analysis The Parsing Problem Recursive-Descent Parsing Bottom-Up Parsing Names, Bindings, and Scopes Names Variables The Concept of Binding Scope Data Types Primitive Data Types Character String Types Enumeration Types Array Types Associative Arrays Expressions and Assignment Statements Arithmetic Expressions and Overloaded Operators Type Conversions Relational and Boolean Expressions Short-Circuit Evaluation Assignment Statements Mixed-Mode Assignment Statement-Level Control Structures Selection Statements Iterative Statements Unconditional Branching Guarded Commands</ li> Subprograms Fundamentals of Subprograms Design Issues for Subprograms Parameter-Passing Methods Design Issues for Functions Implementing Subprograms The General Semantics of Calls and Returns Implementing Simple Subprograms Implementing Dynamic Scoping Abstract Data Types and Encapsulation Constructs The Concept of Abstraction Design Issues for Abstract Data Types Language Examples Encapsulation Constructs Support for Object- Oriented Programming Object-Oriented Programming Design Issues for Object-Oriented Languages Reflection Concurrency Introduction to Subprogram-Level Concurrency Java Threads C# Threads Concurrency in Functional Languages Exception Handling and Event Handling Exception Handling in C++ Exception Handling in Java Exception Handling in Python and Ruby Event Handling with Java Event Handling in C# Functional Programming Languages The First Functional Programming Language: Lisp Common Lisp ML Haskell F# Logic Programming Languages An Overview of Logic Programming The Origins and Basic Elements of Prolog Applications of Logic Programming Relational Database Management Systems Expert Systems Natural-Language Processing

Lab Content

Write a "Hello, World!" program in several programming languages, each representing a different paradigm. Write programs in different languages to declare variables and demonstrate various data types (integers, floats, strings, etc.). Implement conditional statements (if-else) and loops (for, while) in multiple languages to solve a simple problem. Create functions or procedures in different languages to perform basic arithmetic operations. Implement a simple class and objects in an object-oriented language (e.g., Java or C++). Write a program in a functional language (e.g., Scheme or Haskell) to perform a task using recursion and higher-order functions. Solve a logic puzzle using a logic programming language such as Prolog. Analyze and compare two programming languages in terms of syntax, semantics, strengths, and weaknesses. Write a simple lexical analyzer in a language like Python to tokenize a given input code.

Method(s) of Instruction

- Lecture (02)
- DE Live Online Lecture (02S)
- DE Online Lecture (02X)

- Lab (04)
- DE Live Online Lab (04S)
- DE Online Lab (04X)

Instructional Techniques

This course will utilize a combination of lecture, hands-on guided laboratory assignments, classroom/discussion student interactions, problem solving, quizzes, tests, and troubleshooting assignments to achieve the goals and objectives of this course. All instructional methods are consistent across all modalities.

Reading Assignments

Read about the historical development of programming languages and their role in computer science. Read about language design principles, including data abstraction and encapsulation.

Writing Assignments

Analyze and compare programming languages based on their syntax, semantics, and data types. Complete hands-on coding exercises.

Out-of-class Assignments

Evaluate various programming languages for specific tasks to select the appropriate language. Discuss language design principles. Compare programming languages based on their syntax.

Demonstration of Critical Thinking

Students will be asked to read and review academic concepts in the textbook while experiencing how working code behaves under different conditions. Thus, practical programming techniques are taught through personal understanding, hands-on discovery, active learning, and discussing concepts and their results with others.

Required Writing, Problem Solving, Skills Demonstration

The non-mathematical problem-solving exercises will include programming language analysis and comparisons.Programming skills demonstrations are included in the Projects, Midterm and Final examinations.

Textbooks Resources

1. Required Sebesta, R.W. Concepts of Programming Languages, 12th ed. Pearson, 2019 Rationale: - Legacy Textbook Transfer Data: low-cost option

Other Resources

1. Coastline Library 2. Technology related white papers and articles are available at no charge to all students at multiple sites as recommended by the instructor.