Republic of the Philippines
SULTAN KUDARAT STATE UNIVERSITY
Isulan, Sultan Kudarat
Province of Sultan Kudarat
College of Computer Studies

COURSE SYLLABUS IN FUNDAMENTALS OF PROGRAMMING
First Semester, Academic Year 2024-2025

UNIVERSITY VISION
A leading University in advancing scholarly innovation, multi-cultural convergence, and responsive public service in a borderless Region.

UNIVERSITY MISSION
The University shall primarily provide advanced instruction and professional training in science and technology, agriculture, fisheries, education and other relevant fields of study. It shall also undertake research and extension services, and provide progressive leadership in its areas of specialization.

CORE VALUES
Patriotism, **R**espect**, I**ntegrity, **Z**eal, **E**xcellence in Public Service **(PRIZE)**

STRATEGIC GOALS
- ***D***eliver quality service to stakeholders to address current and future needs in instruction, research, extension, and production
- ***O***bserve strict implementation of the laws as well as the policies and regulations of the University.
- ***A***cquire with urgency state-of-the-art resources for its service areas;
- ***B***olster the relationship of the University with its local and international customers and partners.
- ***L***everage the qualifications and competences in personnel action and staffing.
- ***E***valuate the efficiency and responsiveness of the University systems and processes.

Program Objectives and Their Relationships to Strategic Goals:

PROGRAM OBJECTIVES (PO)	**STRATEGIC GOALS**					
A graduate of BS in Computer Science can:	**D**	**O**	**A**	**B**	**L**	**E**
1. innovate technological concepts and ideas underpinning desired IT solutions;	/	/	/	/	/	/
2. administer competently the computer networks, systems development, software applications, hardware, and maintenance;	/	/	/	/	/	/
3. design industry-based applications, infrastructures, and technologies that will promote the advancement and development of the community;	/	/			/	/
4. Adopt to various national and international industries standards in the practice of the profession; and,	/	/	/	/	/	/
5. demonstrate professionalism in the social, environmental, and legal aspects of information technology.	/	/	/	/	/	/

1.	**Course Code**	: CS111
2.	**Course Title**	: Fundamentals of Programming
3.	**Prerequisite**	: None
4.	**Credits**	: 3 Units

5. **Course Description:**

The "Fundamentals of Programming" course serves as an essential foundation for individuals interested in the world of computer programming and software development. Through a comprehensive exploration of programming concepts, this course introduces students to the core principles and practices required to design, write, and execute computer programs effectively.

6. **Course Learning Outcomes and its Relationships to Program Objectives**

COURSE LEARNING OUTCOMES	**PROGRAM OBJECTIVES**				
At the end of the course, a BS Computer Science student can:	a	b	c	d	e
a. Understand the role of programming languages in solving real-world problems;				/	/
b. Develop a solid grasp of basic programming syntax, data types, and control structures;	/	/	/	/	
c. Learn to design and implement structured algorithms using decision-making and looping constructs;	/	/	/	/	/
d. Acquire the skills to create modular programs using functions and manage variables' scope;	/	/	/	/	/
e. Gain proficiency in working with arrays and lists for data storage and manipulation;	/			/	
f. Explore the fundamentals of object-oriented programming (OOP) and class-based structures;		/	/		
g. Develop problem-solving skills through hands-on coding exercises and assignments;					/
h. Learn debugging techniques to identify and resolve common programming errors,	/	/	/	/	/
i. Cultivate an understanding of error handling and the use of exceptions; and					
j. Apply acquired knowledge to create simple programming projects.	/	/	/	/	/

7**. Course Contents**

Course Objectives, Topics, Time Allotment	**Desired Student Learning Outcomes**	**Outcomes-Based Assessment (OBA) Activities**	**Evidence of Outcomes**	**Course Learning Outcomes**	**Program Objectives**	**Values Integration**
Topic 1: ***SKSU VMGO, Classroom Policies, Course Overview, Course Requirements, Grading System (2 hours)***						
1.1. Explain SKSU vision, mission, goals and objectives, classroom policies, course overview, course requirements and grading system.	The students can recall and discuss the SKSU vision, mission, goals and objectives, classroom policies, course overview, course requirements and grading system.	Lecture Group Works		a, b, c, d, e, f	a, d, e	Open-mindedness, security, self-control and Inclusiveness
Topic 2: ***Introduction to Programming Concepts and Environment (12 Hours)***						
2.1. Understanding the role of programming in problem-solving 2.2. Introduction to programming languages and their classifications 2.3. Setting up the programming environment (IDEs, compilers, etc.) 2.4. Writing and running a simple "Hello, World!" program 2.5. Basic syntax, variables, and data types	By the end of this topics, students will be able to: Recognize how programming serves as a tool for designing solutions and automating tasks. Formulate problem-solving strategies that leverage programming concepts effectively. Differentiate between various programming languages and their intended applications. Categorize programming languages based on their paradigms and use cases. Independently configure integrated development environments (IDEs) to suit project needs.	Research and Presentation Case Study Analysis Group Discussion Reflection Paper	The students provide written explanations or participate in discussions where they describe real-world problems that programming can address. They showcase how programming solutions can automate tasks, improve efficiency, or solve complex challenges.	a, b, c, d	a, b, c, e	Excellence, Professional resourcefulness and leadership

	Install and set up compilers, interpreters, and related tools for selected programming languages. Navigate and utilize essential features of the chosen development environment effectively. Independently compose a basic program that displays "Hello, World!" on the screen. Grasp the fundamental structure of a program, including headers and main functions. Define and manipulate variables to store and manage data during program execution. Identify and utilize common data types, such as integers, strings, and floats, appropriately.		The students submit a series of code snippets demonstrating their understanding of basic syntax rules, variable declaration and initialization, and the usage of different data types (integers, strings, floats). They provide annotations explaining each snippet's functionality.			
Topic 3: ***Control Structures and Decision Making (10 Hour)***						
3.1 Introduction to control structures: if, else if, else statements 3.2 Logical operators and their use in decision making 3.3 Switch statements for multi-way branching 3.4 Introduction to loops: while and for loops 3.5 Loop control statements: break and continue	By the end of this topics, students will be able to: Comprehend the purpose of control structures in programming for making decisions. Skillfully implement if statements to execute code based on specific conditions.	Algorithm Comparison Model Evaluation Unsupervised Learning Project	The students demonstrate their understanding by writing code that includes if, else if, and else statements to make decisions based on	b, c, d, e, f	a, b, c, d, e	Knowledge, Innovation, Quality and Recognition

	Articulate the concept of else if and else statements for multi-alternative decisions. Master the logical operators (AND, OR, NOT) and their role in evaluating conditions. Apply logical operators to formulate complex conditions for decision-making. Utilize logical operators effectively to create efficient and accurate control structures. Explain the function of switch statements in handling multiple branching scenarios. Develop switch statements to streamline decision-making when several choices are present. Evaluate when switch statements are more suitable than multiple if-else constructs. Grasp the significance of loops for executing code repetitively. Construct while loops to repeatedly execute code while a condition holds true. Implement for loops to perform iterations for a predetermined number of times.	Evaluation Metrics Discussion	different conditions. They explain their code choices and how these control structures address various scenarios. The students provide code segments that utilize loop control statements, specifically break and continue, to manage the flow of loops. They describe the situations in which these statements are used and how they impact loop execution.			

	Understand the purposes of loop control statements, namely break and continue. Apply the break statement to exit loops prematurely when a specific condition is met. Employ the continue statement to skip to the next iteration of a loop under certain conditions.					
Topic 4. ***Functions and Modularization (15 Hour)***						
4.1. Defining and using functions 4.2. Function parameters and return values 4.3. Scope and lifetime of variables 4.4. Introduction to modular programming 4.5. Creating and using libraries/modules	By the end of this topics, students will be able to: Articulate the purpose and advantages of functions in programming. Formulate functions with clear names and meaningful documentation. Develop a library of functions to modularize code and enhance reusability. Grasp the concept of function parameters and their role in accepting input. Design functions with parameters to process and utilize external data. Expertly employ return statements to convey computed results back to the calling code.	Lecture, Group Discussion on Assigned Topics Research assignment	The students present code snippets featuring self-defined functions that perform specific tasks. They explain how functions help modularize code and enhance reusability, showcasing their ability to break down complex logic into manageable units.	b, c, d, e, f	a, b, c, d, e	Flexibility, Mindfulness, Resilience, Patience and Personal Development

	Understand variable scope and its impact on variable accessibility. Identify local and global variables and manage their usage effectively. Control variable lifetimes to optimize memory utilization and prevent leaks. Recognize the significance of modular programming in fostering code organization. Deconstruct complex tasks into smaller, manageable modules. Integrate modules to create cohesive and maintainable code structures. Comprehend the role of libraries/modules in encapsulating reusable code components. Construct custom libraries/modules to bundle related functionalities effectively. Incorporate external libraries/modules to extend programming capabilities and efficiency.		The students showcase their comprehension by presenting code that exemplifies modular programming. They discuss how modular design enhances code organization, reusability, and collaboration by breaking down complex tasks into manageable modules.			

Topic 5: ***Arrays and Lists (15 Hours)***						
5.1. Introduction to arrays and their uses 5.2. Declaring, initializing, and accessing array elements 5.3. Array traversal techniques: for-each and traditional for loop 5.4. Introduction to lists and their advantages 5.5. Basic list operations: append, insert, remove	By the end of this topics, students will be able to: Understand the concept of arrays as data structures for storing multiple elements. Identify scenarios where arrays are utilized to organize related data efficiently. Discuss the benefits and limitations of using arrays in programming. Master the syntax for declaring arrays and specifying their data types. Proficiently initialize arrays with appropriate values during declaration. Access individual elements within arrays using index-based referencing. Compare and contrast traversal techniques: for-each loop and traditional for loop. Utilize for-each loops to iterate through arrays and process elements. Employ traditional for loops to control the iteration process with more flexibility.	Class Discussion Research assignment	The students provide explanations or code examples that demonstrate their understanding of arrays as data structures for storing multiple elements. They discuss scenarios where arrays are beneficial, showcasing their awareness of how arrays organize related data efficiently. The students provide code snippets that showcase their ability to perform basic list operations, including appending, inserting, and removing elements. They	b, c, d, e, f	a, b, c	Boldness, Compassion, Brilliance, Intelligence, Preparedness, Acceptance and Making a Difference

	Define lists as dynamic data structures capable of handling varying numbers of elements. Enumerate the advantages of using lists over arrays in certain scenarios. Recognize situations where lists provide more efficient data management solutions. Implement the append operation to add elements to the end of a list. Utilize insert to add elements at specific positions within a list. Perform the remove operation to eliminate elements from a list while maintaining integrity.		explain the reasons for using each operation and how these actions contribute to effective data manipulation.			
Topic 6: ***Object-Oriented Programming (OOP) Basics (15 Hours)***						
6.1. Introduction to object-oriented programming 6.2. Classes and objects: definition and instantiation 6.3. Class properties (attributes) and methods (functions) 6.4. Encapsulation and information hiding 6.5. Introduction to constructors and destructors	By the end of this topics, students will be able to: Grasp the fundamental concepts of object-oriented programming (OOP). Differentiate between procedural and OOP paradigms. Recognize the significance of OOP in creating organized and modular code. Define the concept of classes as blueprints for creating objects.	Class Discussion	The students provide written explanations or presentations that showcase their understanding of object-oriented programming (OOP) concepts. They explain the shift from procedural to OOP	b, c, d, e, f	a, b, c	Truthfulness, Patience, Knowledge

	Construct classes with proper naming and a clear understanding of their purpose. Create instances (objects) of classes to represent real-world entities or concepts. Understand class properties (attributes) as characteristics associated with objects. Implement class methods (functions) to define behavior and actions. Distinguish between instance attributes/methods and class attributes/methods. Comprehend the concept of encapsulation in OOP as bundling data and methods. Utilize access modifiers (public, private, protected) to control data visibility. Apply information hiding principles to promote data integrity and code maintainability. Define constructors as special methods used to initialize objects. Create constructors to set initial values for object properties.		paradigms and how OOP promotes code organization and modularity. The students provide written or code-based explanations that demonstrate their understanding of encapsulation. They discuss how encapsulation bundles data and methods, and they explain the role of access modifiers (public, private, protected) in controlling data visibility.			

	Explore the concept of destructors to manage resources and perform cleanup tasks.					
Topic 7: ***Error Handling and Debugging (10 Hours)***						
7.1. Types of errors: syntax, runtime, and logic errors 7.2. Debugging techniques and strategies 7.3. Using debugging tools and IDE features 7.4. Introduction to exception handling 7.5. try, catch, and finally blocks for exception handling	By the end of this topics, students will be able to: Recognize the distinctions between syntax, runtime, and logic errors. Identify the causes and manifestations of each error type. Describe the significance of effective error handling in programming. Develop proficiency in identifying and isolating errors within code. Apply systematic debugging strategies, such as binary search and divide-and-conquer. Utilize strategic thinking to troubleshoot and rectify errors efficiently. Navigate and utilize debugging tools within integrated development environments (IDEs). Employ breakpoints, watches, and step-by-step execution for effective debugging.	Talking Circle and Reporting Teachers' Lecture	The students provide written explanations or code examples that showcase their ability to differentiate between syntax, runtime, and logic errors. They discuss the causes and effects of each error type, highlighting their understanding of common programming mistakes. The students showcase their proficiency by providing screenshots or written explanations of their utilization of debugging tools within	b, c, d, e, f	a, b, c	Caring, Ethics, Security, Excellence and Consistency

	Leverage IDE features to inspect variables, trace program flow, and pinpoint issues. Understand the concept of exceptions and their role in handling unexpected scenarios. Differentiate between exceptions and regular errors in programming. Appreciate the significance of graceful program behavior in the face of exceptions. Construct try blocks to encapsulate potentially exception-raising code. Develop catch blocks to capture and handle specific exception types. Implement finally blocks for cleanup tasks regardless of exception occurrence.		integrated development environments (IDEs). They describe how they use breakpoints, watches, and step-by-step execution to troubleshoot code effectively.			
Topic 8: ***Final Projects and Wrap-Up (15 Hours)***						
8.1. Applying concepts learned to create a simple programming project 8.2. Review of major topics and concepts covered throughout the course 8.3. Discussion on further programming studies and career opportunities	By the end of this topics, students will be able to: Apply acquired knowledge and skills to design and develop a basic programming project. Utilize appropriate programming concepts to address specific project requirements.	Group Reports	The students present their completed programming project that integrates concepts learned throughout the course. They showcase how they've applied	b, c, d, e, f	a, b, c, d, e	Preparedness, Decisiveness, Credibility, and Ethics

	Demonstrate creativity and problem-solving abilities in project execution. Summarize and reflect on the key concepts and principles studied throughout the course. Consolidate understanding of programming fundamentals, control structures, OOP, and more. Integrate knowledge from different modules to build a holistic perspective on programming. Organize and revise course materials, notes, and assignments for effective exam preparation. Review key topics, practice problem-solving, and apply critical thinking to exam-style questions. Exhibit confidence in approaching the final exam by demonstrating mastery of course content. Explore potential career paths and opportunities in programming and related fields. Engage in conversations about further studies, specializations, and advanced programming concepts. Make informed decisions about future learning paths based on course experiences and interests.		foundational skills to design, develop, and execute a functional program that addresses a specific problem or task. The students engage in written discussions or presentations where they explore potential programming study paths and career trajectories. They demonstrate awareness of specialized areas of interest, post-course learning opportunities, and potential career roles that align with their skills and interests.			

8. **Course Evaluation**

Course Requirements	1. Regular Attendance and Active Participation: Attend all classes and actively engage in discussions, exercises, and group activities. 2. Assignments and Exercises: Completion of regular assignments and exercises to reinforce understanding of the course material. Assignments may include programming tasks, problem-solving exercises, and code debugging. 3. Project Work: Undertake programming projects to apply concepts learned in real-world scenarios. Projects should involve the application of basic programming constructs and problem-solving techniques. 4. Presentations: Present project outcomes to the class, demonstrating the functionality and logic of the implemented solutions. 5. Examinations: Participate in periodic quizzes or exams to assess comprehension of fundamental programming concepts and techniques. 6. Readings and Literature Review: Read relevant textbook chapters and resources to deepen understanding of programming principles. Potentially write brief summaries or reflections on assigned readings. 7. Class Discussions and Participation: Actively contribute to class discussions, asking questions, sharing insights, and engaging in topic-related debates. 8. Ethical Considerations Assignment: Complete an assignment exploring ethical considerations in programming. Analyze potential ethical issues related to software development and propose ethical coding practices. 9. Final Assessment: A comprehensive final assessment covering key programming concepts and techniques learned throughout the course. May include a combination of written questions and coding challenges.
Course Policies	1. Attendance Policy: Students are expected to attend all classes and arrive on time. Regular attendance is crucial for active participation and engagement in the course. Any absences should be communicated in advance, if possible. 2. Grading Policy: Clear grading criteria should be provided to students at the beginning of the course, outlining the weightage of different assessments such as assignments, exams, projects, and class participation. The grading policy should be fair and transparent. 3. Late Submission Policy: Assignments and projects should have specific deadlines, and late submissions may incur penalties. The policy for late submissions should be clearly communicated to students, including any deduction in marks or potential restrictions on late submissions. 4. Academic Integrity Policy: Students should adhere to a strict academic integrity policy. Plagiarism, cheating, or any form of academic dishonesty will not be tolerated and may result in disciplinary actions. Guidelines on proper citation and referencing should be provided. 5. Collaboration Policy: Collaboration among students should be encouraged, but it is essential to clarify the boundaries and expectations regarding collaboration on assignments and projects. Clearly define what is permissible and what is considered unauthorized collaboration. 6. Technology and Equipment Policy: If specific software or hardware tools are required for the course, ensure that students have access to them or provide alternatives. Communicate any technology requirements and policies regarding the use of personal devices in class. 7. Communication Policy: Establish clear channels of communication between the instructor and students, such as email or a learning management system (LMS). Response time for queries should be communicated, along with guidelines for appropriate communication. 8. Inclusivity and Respect Policy: Foster an inclusive and respectful learning environment where all students are treated with dignity. Encourage open-mindedness, active listening, and constructive feedback among students. 9. Accommodation Policy: Provide accommodations for students with disabilities or specific needs, as per institutional policies. Students should be encouraged to communicate their needs and make necessary arrangements in consultation with the relevant authorities.

	10. Revision Policy: Clarify the policy for requesting revisions or feedback on assignments, exams, or projects. Specify any limitations or deadlines for requesting revisions.	
Grading System	• Assignments and Exercises: 20% ○ Regular assignments and exercises throughout the course. • Project Work: 20% ○ The project involves applying fundamentals of programming techniques to a real-world problem. • Examinations: 40% ○ Midterm and final exams to assess understanding of the theoretical concepts, algorithms, and techniques covered in the course. • Class Participation: 10% ○ Actively engaging in class discussions, asking questions, and contributing to the learning environment. • Ethical Considerations Assignment: 10% ○ Completion of an assignment exploring the ethical implications and considerations associated with the development of project.	***Class Schedule:*** Schedule of Examination: Refer to the university calendar

References:

Textbooks	1. "Starting Out with Programming Logic and Design" by Tony Gaddis and Godfrey Muganda 2. "Python Programming: An Introduction to Computer Science" by John Zelle 3. "C++ Programming: From Problem Analysis to Program Design" by D.S. Malik 4. "Java Programming: From Problem Analysis to Program Design" by D.S. Malik 5. "Programming in C" by Stephen G. Kochan 6. "Introduction to Java Programming and Data Structures" by Y. Daniel Liang 7. "C# Programming Yellow Book" by Rob Miles 8. "C Programming Absolute Beginner's Guide" by Perry, Miller, and Miller 9. "Introduction to the Theory of Computation" by Michael Sipser (for a deeper understanding of computational theory) 10. Sams Teach Yourself C++ in 24 Hours by Jesse Liberty · 2002
Online References	1. Learn C++ – Skill up with our free tutorials (learncpp.com) 2. cplusplus.com/doc/tutorial/ 3. C++ Programming Language - GeeksforGeeks 4. https://www.tutorialspoint.com/cplusplus/index.htm 5. Top C Courses - Learn C Online | Coursera 6. Learn C++ with Online Courses and Programs | edX 7. https://www.codecademy.com/ 8. https://www.w3schools.com/cpp/ 9. https://www.softwaretestinghelp.com/cpp-tutorials/ 10. Learn C++ – Skill up with our free tutorials (learncpp.com)

Prepared by:	Reviewed by:	Approved by:
ELBREN O. ANTONIO, DIT Subject Professor	**CECILIA E. GENER, PhD** Chairman, BSCS	**BENNEDICT A. RABUT, DIT** Dean, College of Computer Studies