

NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF MATHEMATICAL SCIENCES

SYLLABUS

1. GENERAL INFORMATION

1.1 Course name	:	Introduction to Programming
1.2 Code	:	CS111
1.3 Academic Program	:	Scientific Computing
1.4 Credits	:	5
1.5 Prerrequisites	:	None
1.6 Type of Course	:	Mandatory
1.7 Area of Studies	:	Estudios Generales
1.8 Number of Weeks	:	16
1.9 Hours per week	:	3 HT; 4 HP;
1.10 Cycle	:	I
1.11 Learning modality	:	Face to face
1.12 Academic Period	:	2026-I
1.13 Semester Start Date	:	
1.14 Semester End Date	:	

2. PROFESSORS

Meetings after coordination with the professor

3. INTRODUCTION TO THE COURSE

This is the first course in the sequence of introductory courses to Computer Science. This course introduces participants to the fundamental concepts of programming using Python. Topics include data types, control structures, functions, lists, recursion, and the mechanics of execution, testing, and debugging. Programming is one of the pillars of Computer Science; any professional in the area will need to program to materialize their models and proposals.

4. GOALS

- Introduce the fundamental concepts of programming.
- Develop the ability of abstraction using a programming language.
- Develop problem-solving skills through algorithmic thinking.
- Understand basic data structures and their applications.
- Master the Python programming language for problem solving.

5. COMPETENCES

ABET-1) An ability to identify, formulate, and solve broadly defined technical or scientific problems by applying knowledge of mathematics and science and/or technical topics to areas relevant to the discipline. (Familiarity)

ABET-2) An ability to formulate or design a system, process, procedure, or program for the intended purpose. (Usage)

6. TOPICS

Unit 1: Introduction to Programming and History (4 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Brief history of computing and programming languages. • Overview of Python programming language. • Programming environments and tools. • Basic program structure and execution. • Writing, running, and debugging simple programs.	• Identify important trends in the history of computing. [Familiarizarse] • Discuss the historical context of programming language paradigms. [Familiarizarse] • Compare daily life before and after personal computers and the Internet. [Evaluar] • Write, compile, and run a simple Python program. [Usar]
Readings : [**Zelle2010**], [**Guttag2013**], [**Brooks2019**]	

Unit 2: Type Systems Fundamentals (6 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• A type as a set of values together with a set of operations: – Primitive types (e.g., numbers, Booleans) – Compound types built from other types (e.g., records/structs, unions, arrays, lists, functions, references using set operations) • Type safety and errors caused by using values inconsistently given their intended types • Goals and limitations of static and dynamic typing: detecting and eliminating errors as early as possible. • Primitive data types in Python (int, float, str, bool). • Type conversion and casting.	• Describe, for both a primitive and a compound type, the values that have that type [Familiarizarse] • Describe examples of program errors detected by a type system [Familiarizarse] • Use types and type-error messages to write and debug programs [Usar] • Explain Python's dynamic typing system. [Familiarizarse]
Readings : [**Zelle2010**], [**Guttag2013**]	

Unit 3: Object-Oriented Programming Fundamentals (14 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Imperative programming as a subset of object-oriented programming. • Definition of classes: fields, methods, and constructors. • Dynamic vs static properties. • Variables, expressions, and assignments. • Input and output operations. • Conditional statements (if, elif, else). • Iteration structures (for loops, while loops). • Nested loops and loop control statements. • Exception handling basics.	• Compose a class through design, implementation, and testing to meet behavioral requirements [Usar] • Predict and validate control flow in a program using dynamic dispatch [Evaluar] • Use object-oriented encapsulation mechanisms such as interfaces and private members [Usar] • Implement programs using conditional and iterative structures. [Usar] • Debug programs with syntax and logical errors. [Usar] • Design and implement simple algorithms using loops. [Usar]
Readings : [**Zelle2010**], [**Guttag2013**], [**Brooks2019**]	

Unit 4: Functions and Parameter Passing (10 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Function definition and invocation. • Parameter passing mechanisms (by value, by reference). • Return values and multiple returns. • Variable scope and lifetime (local, global, nonlocal). • Recursion fundamentals. • Lambda expressions and higher-order functions. • Functions as first-class objects. • Documentation and testing of functions.	• Define and implement functions with proper parameter passing. [Usar] • Explain the difference between pass-by-value and pass-by-reference. [Familiarizarse] • Implement recursive solutions to simple problems. [Usar] • Use lambda expressions for functional programming constructs. [Usar] • Design functions with appropriate scope and documentation. [Evaluar]
Readings : [**Zelle2010**], [**Guttag2013**], [**Pragmatic2009**]	

Unit 5: Basic Data Structures (10 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Lists and list operations (indexing, slicing, mutability). • Tuples and immutability. • Dictionaries and key-value mapping. • Sets and set operations. • Strings as sequences. • List comprehensions and generator expressions. • Basic searching and sorting algorithms. • Introduction to algorithmic complexity for basic operations.	• Implement programs using lists, dictionaries, and sets. [Usar] • Choose appropriate data structures for given problems. [Evaluar] • Explain the trade-offs between different data structures. [Familiarizarse] • Implement basic search and sort operations on lists. [Usar]
Readings : [**Zelle2010**], [**Guttag2013**]	

Unit 6: Object-Oriented Programming Fundamentals (10 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Subclasses, inheritance (including multiple inheritance), and method overriding. • Dynamic dispatch: definition of method-call. • Object-oriented idioms for encapsulation: – Privacy, data hiding, and visibility of class members. – Interfaces revealing only method signatures. – Abstract base classes, traits and mixins. • Classes and objects in Python. • Attributes and methods. • Inheritance and polymorphism basics. • Encapsulation and information hiding.	• Build a simple class hierarchy utilizing subclassing that allows code to be reused for distinct subclasses [Usar] • Explain the relationship between object-oriented inheritance (code-sharing and overriding) and subtyping (the idea of a subtype being usable in a context that expects the supertype) [Familiarizarse] • Compare and contrast the benefits and costs/impact of using inheritance (subclasses) and composition (specifically, how to base composition on higher order functions) [Evaluar] • Design and implement simple classes in Python. [Usar] • Explain the benefits of OOP for code organization. [Familiarizarse]
Readings : [**Zelle2010**], [**Guttag2013**]	

Unit 7: Algorithmic Strategies (6 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Paradigms: – Brute-Force (e.g., linear search, selection sort, traveling salesperson, knapsack) – Decrease-and-Conquer: ∗ By a Constant (e.g., insertion sort, topological sort) ∗ By a Constant Factor (e.g., binary search) ∗ By a Variable Size (e.g., Euclid's) – Divide-and-Conquer (e.g., binary search, quicksort, mergesort, Strassen's) – Greedy (e.g., Dijkstra's, Kruskal's, Knapsack) – Transform-and-Conquer: ∗ Instance simplification (e.g., find duplicates via list presort) ∗ Representation change (e.g., heapsort) ∗ Problem reduction (e.g., least-common-multiple, linear programming) ∗ Dynamic programming (e.g., Floyd's, Marshall, Bellman-Ford) – Space vs time tradeoffs (e.g., hashing) • Iteration vs recursion (e.g., factorial, tree search) • Problem decomposition strategies. • Algorithm design techniques. • Debugging and testing strategies. • Tracing program execution.	• For each of the paradigms in this unit: – Explain its definitional characteristics – Explain an example that demonstrates the paradigm including how this example satisfies the paradigm's characteristics. [Familiarizarse] • Give examples of iterative and recursive algorithms that solve the same problem, explain the benefits and disadvantages of each approach [Usar] • Evaluate whether a greedy approach leads to an optimal solution [Evaluar] • Apply systematic debugging techniques to locate and fix errors. [Usar] • Analyze algorithm complexity for simple problems. [Familiarizarse]
Readings : [**Guttag2013**], [**Cormen2009**]	

Unit 8: Modern Development Methods and Tool Usage (4 hours)	
Competences Expected: ABET-1,ABET-2	
Topics	**Learning Outcomes**
• Using modern programming environments (IDEs, notebooks). • Code search and reuse best practices. • Using standard libraries and APIs. • Debugging and testing techniques. • Version control basics. • Code documentation and style guidelines. • Virtual environments and package management.	• Use modern development tools effectively. [Usar] • Apply code reuse principles through libraries and APIs. [Usar] • Debug and test programs systematically. [Usar] • Follow coding standards and documentation practices. [Familiarizarse] • Manage Python packages and environments. [Familiarizarse]
Readings : [**Zelle2010**], [**Pragmatic2009**]	

7. WORKPLAN

7.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

7.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

7.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

8. EVALUATION SYSTEM

********* EVALUATION MISSING ********

9. BASIC BIBLIOGRAPHY