San Pablo Catholic University (UCSP)
Undergraduate Program in
Computer Science
SILABO

CS111. Videogames Programming (Mandatory)

2023-II

1. General information

1.1 School	:	Ciencia de la Computación
1.2 Course	:	CS111. Videogames Programming
1.3 Semester	:	1^{er} Semestre.
1.4 Prerrequisites	:	None
1.5 Type of course	:	Mandatory
1.6 Learning modality	:	Face to face
1.7 Horas	:	2 HT; 4 HP;
1.8 Credits	:	4
1.9 Plan	:	Plan Curricular 2016

2. Professors

Lecturer

- Yessenia Deysi Yari Ramos <ydyari@ucsp.edu.pe>
  - PhD in Computer Science, UNSA, PERU, 2022.
  - MSc in Ciencias de la Computación, UFRGS, Brasil, 2011.
- Kelly Vizconde la Motta <kvizconde@ucsp.edu.pe>
  - MSc in Mag. Ciencia de la Computación, Universidad Católica San Pablo, Perú, 2019.

3. Course foundation

This is the first course in the sequence of introductory courses to Computer Science.This course is intended to cover the concepts outlined by the Computing Curricula IEEE-CS/ACM 2013. Programming is one of the pillars of Computer Science; any professional of the area, will need to program to materialize their models and proposals. This course introduces participants to the fundamental concepts of this art. Topics include data types, control structures, functions, lists, recursion, and the mechanics of execution, testing, and debugging.

4. Summary

1. History 2. Basic Type Systems 3. Fundamental Programming Concepts 4. Basic Analysis 5. Fundamental Data Structures and Algorithms 6. Algorithms and Design 7. Development Methods

5. Generales Goals

- Introduce the fundamental concepts of programming.
- Develop the ability of abstraction using programming language

6. Contribution to Outcomes
This discipline contributes to the achievement of the following outcomes: **1)** Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions. (**Usage**) **2)** Design, implement and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline. (**Usage**) **6)** Apply computer science theory and software development fundamentals to produce computing-based solutions. (**Usage**)

7. Content

UNIT 1: History (5)	
Competences:	
Content	**Generales Goals**
• Prehistory, the world before 1946 • History of computer hardware, software, networking • Pioneers of computing • History of the Internet	• Identify significant continuing trends in the history of the computing field [Familiarity] • Identify the contributions of several pioneers in the computing field [Familiarity] • Discuss the historical context for several programming language paradigms [Familiarity] • Compare daily life before and after the advent of personal computers and the Internet [Assessment]
Readings: Brookshear2019, Guttag (2013), Zelle (2010)	

UNIT 2: Basic Type Systems (2)	
Competences:	
Content	**Generales Goals**

• 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, unions, arrays, lists, functions, references)
• Association of types to variables, arguments, results, and fields
• Type safety and errors caused by using values inconsistently given their intended types
	• For both a primitive and a compound type, informally describe the values that have that type [Familiarity]
	• For a language with a static type system, describe the operations that are forbidden statically, such as passing the wrong type of value to a function or method [Familiarity]
	• Describe examples of program errors detected by a type system [Familiarity]
	• For multiple programming languages, identify program properties checked statically and program properties checked dynamically [Usage]
	• Use types and type-error messages to write and debug programs [Usage]
	• Define and use program pieces (such as functions, classes, methods) that use generic types, including for collections [Usage]

Readings: Guttag (2013), Zelle (2010)	

UNIT 3: Fundamental Programming Concepts (9)	
Competences:	
Content	**Generales Goals**
• Basic syntax and semantics of a higher-level language • Variables and primitive data types (e.g., numbers, characters, Booleans) • Expressions and assingments • Simple I/O including file I/O • Conditional and iterative control structures • Functions and parameter passing • The concept of recursion	• Analyze and explain the behavior of simple programs involving the fundamental programming constructs variables, expressions, assignments, I/O, control constructs, functions, parameter passing, and recursion. [Assessment] • Identify and describe uses of primitive data types [Familiarity] • Write programs that use primitive data types [Usage] • Modify and expand short programs that use standard conditional and iterative control structures and functions [Usage] • Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, the definition of functions, and parameter passing [Usage] • Write a program that uses file I/O to provide persistence across multiple executions [Usage] • Choose appropriate conditional and iteration constructs for a given programming task [Familiarity] • Describe the concept of recursion and give examples of its use [Assessment] • Identify the base case and the general case of a recursively-defined problem [Familiarity]
Readings: Guttag (2013), Zelle (2010)	

UNIT 4: Basic Analysis (2)	
Competences:	
Content	**Generales Goals**
• Differences among best, expected, and worst case behaviors of an algorithm • Big O notation: formal definition • Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential • Big O notation: use • Analysis of iterative and recursive algorithms	• Explain what is meant by "best", "expected", and "worst" case behavior of an algorithm [Familiarity] • In the context of specific algorithms, identify the characteristics of data and/or other conditions or assumptions that lead to different behaviors [Familiarity] • State the formal definition of big O [Familiarity] • Use big O notation formally to give asymptotic upper bounds on time and space complexity of algorithms [Usage] • Use big O notation formally to give expected case bounds on time complexity of algorithms [Usage]
Readings: Guttag (2013), Zelle (2010)	

UNIT 5: Fundamental Data Structures and Algorithms (8)	
Competences:	
Content	**Generales Goals**

• Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max,
• Sequential and binary search algorithms
• Worst case quadratic sorting algorithms (selection, insertion)
• Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, mergesort)
• Hash tables, including strategies for avoiding and resolving collisions
• Binary search trees
  – Common operations on binary search trees such as select min, max, insert, delete, iterate over tree
• Graphs and graph algorithms
  – Representations of graphs (e.g., adjacency list, adjacency matrix)
  – Depth- and breadth-first traversals
• Heaps
• Graphs and graph algorithms
  – Maximum and minimum cut problem
  – Local search
• Pattern matching and string/text algorithms (e.g., substring matching, regular expression matching, longest common subsequence algorithms)
	• Implement basic numerical algorithms [Usage]
	• Implement simple search algorithms and explain the differences in their time complexities [Assessment]
	• Be able to implement common quadratic and O(N log N) sorting algorithms [Usage]
	• Describe the implementation of hash tables, including collision avoidance and resolution [Familiarity]
	• Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing [Familiarity]
	• Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data [Familiarity]
	• Explain how tree balance affects the efficiency of various binary search tree operations [Familiarity]
	• Solve problems using fundamental graph algorithms, including depth-first and breadth-first search [Usage]
	• Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context [Assessment]
	• Describe the heap property and the use of heaps as an implementation of priority queues [Familiarity]
	• Solve problems using graph algorithms, including single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm [Usage]
	• Trace and/or implement a string-matching algorithm [Usage]

Readings: Guttag (2013), Zelle (2010)	

<table>
<tr><td colspan="2">UNIT 6: Algorithms and Design (9)</td></tr>
<tr><td colspan="2">Competences:</td></tr>
<tr><td>Content</td><td>Generales Goals</td></tr>
<tr><td>

The concept and properties of algorithms
Informal comparison of algorithm efficiency (e.g., operation counts)
The role of algorithms in the problem-solving process
Problem-solving strategies
Iterative and recursive mathematical functions
Iterative and recursive traversal of data structures
Divide-and-conquer strategies
Fundamental design concepts and principles
Abstraction
Program decomposition
Encapsulation and information hiding
Separation of behavior and implementation

</td><td>

Discuss the importance of algorithms in the problem-solving process [Familiarity]
Discuss how a problem may be solved by multiple algorithms, each with different properties [Familiarity]
Create algorithms for solving simple problems [Usage]
Use a programming language to implement, test, and debug algorithms for solving simple problems [Usage]
Implement, test, and debug simple recursive functions and procedures [Usage]
Determine whether a recursive or iterative solution is most appropriate for a problem [Assessment]
Implement a divide-and-conquer algorithm for solving a problem [Usage]
Apply the techniques of decomposition to break a program into smaller pieces [Usage]
Identify the data components and behaviors of multiple abstract data types [Usage]
Implement a coherent abstract data type, with loose coupling between components and behaviors [Usage]
Identify the relative strengths and weaknesses among multiple designs or implementations for a problem [Assessment]

</td></tr>
<tr><td colspan="2">Readings: Guttag (2013), Zelle (2010)</td></tr>
</table>

<table>
<tr><td colspan="2">UNIT 7: Development Methods (1)</td></tr>
<tr><td colspan="2">Competences:</td></tr>
<tr><td>Content</td><td>Generales Goals</td></tr>
<tr><td>

Modern programming enviroments
Code search
Programming using library components and their APIs

</td><td>

Construct and debug programs using the standard libraries available with a chosen programming language [Familiarity]

</td></tr>
<tr><td colspan="2">Readings: Guttag (2013), Zelle (2010)</td></tr>
</table>

8. Methodology

1. El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.
2. El profesor del curso presentará demostraciones para fundamentar clases teóricas.
3. El profesor y los alumnos realizarán prácticas

4. Los alumnos deberán asistir a clase habiendo leído lo que el profesor va a presentar. De esta manera se facilitará la comprensión y los estudiantes estarán en mejores condiciones de hacer consultas en clase.

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

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.

Evaluation System:
The final grade is obtained through of:

CONTINUOUS ASSESMENT	EVALUATIONS
Continuous assessment 1 : 24 % **Continuous assessment 2** : 36 %	**Midterm Exam** : 20 % **Final Exam** : 20 %
60%	40%

Where:

Continuous Assessment: It includes group work, active participation in class, exercise test.

- Continuos assessment 1 (weeks 1 - 9)
- Continuos assesment 2 (weeks 10 - 17)

To pass the course you must obtain 11.5 or more in the final grade .

References

Guttag, John V (2013). . *Introduction To Computation And Programming Using Python*. MIT Press.
Zelle, John (2010). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates Inc.