

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



CS341. Programming languages (Mandatory)

1. General information

1.1 School	:	Ciencia de la Computación
1.2 Course	:	CS341. Programming languages
1.3 Semester	:	7 ^{mo} Semestre.
1.4 Prerequisites	:	CS211. Computer Science Theory. (4 th Sem)
1.5 Type of course	:	Mandatory
1.6 Learning modality	:	Virtual
1.7 Horas	:	2 HT; 2 HP; 2 HL;
1.8 Credits	:	4

2. Professors

3. Course foundation

Los lenguajes de programación son el medio a través del cual los programadores describen con precisión los conceptos, formulan algoritmos y representan sus soluciones. Un científico de la computación trabajará con diferentes lenguajes, por separado o en conjunto. Los científicos de la computación deben entender los modelos de programación de los diferentes lenguajes, tomar decisiones de diseño basados en el lenguaje de programación y sus conceptos. El profesional a menudo necesitará aprender nuevos lenguajes y construcciones de programación y debe entender los fundamentos de como las características del lenguaje de programación están definidas, compuestas e implementadas. El uso eficaz de los lenguajes de programación y la apreciación de sus limitaciones, también requiere un conocimiento básico de traducción de lenguajes de programación y su análisis de ambientes estáticos y dinámicos, así como los componentes de tiempo de ejecución tales como la gestión de memoria, entre otros detalles de relevancia.

4. Summary

1. 2. Language Pragmatics 3. Type Systems 4. Object-Oriented Programming 5. Functional Programming 6. Event-Driven and Reactive Programming 7. Logic Programming

5. Generales Goals

- Capacitar a los estudiantes para entender los lenguajes de programación desde diferentes tipos de vista, según el modelo subyacente, los componentes fundamentales presentes en todo lenguaje de programación y como objetos formales dotados de una estructura y un significado según diversos enfoques.

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**)
- 6) Apply computer science theory and software development fundamentals to produce computing-based solutions. (**Usage**)

7. Content

UNIT 1: (18)	
Competences:	
Content	Generales Goals
<ul style="list-style-type: none"> • Historia de los Lenguajes de Programación • Programs that take (other) programs as input such as interpreters, compilers, type-checkers, documentation generators • Data structures to represent code for execution, translation, or transmission • Estructura de un programa: Léxico, Sintáctico y Semántico • BNF • Interpretation vs. compilation to native code vs. compilation to portable intermediate representation [Familiarity] 	<ul style="list-style-type: none"> • Reconocer el desarrollo histórico de los lenguajes de programación. [Familiarity] • Identificar los paradigmas que agrupan a la mayoría de lenguajes de programación existentes hoy en día. [Familiarity] • Explain how programs that process other programs treat the other programs as their input data [Familiarity] • Describe an abstract syntax tree for a small language [Familiarity] • Write a program to process some representation of code for some purpose, such as an interpreter, an expression optimizer, or a documentation generator [Usage] • Distinguish a language definition (what constructs mean) from a particular language implementation (compiler vs interpreter, run-time representation of data objects, etc) [Familiarity] • Reconocer como funciona un programa a nivel de computador. [Familiarity]
Readings: Sebesta (2012), Webber (2010)	

UNIT 2: Language Pragmatics (12)	
Competences:	
Content	Generales Goals
<ul style="list-style-type: none"> • Principles of language design such as orthogonality • Evaluation order, precedence and associativity • Eager vs. delayed evaluation • Defining control and iteration constructs • External calls and system libraries 	<ul style="list-style-type: none"> • Discuss the role of concepts such as orthogonality and well-chosen defaults in language design [Usage] • Use crisp and objective criteria for evaluating language-design decisions [Usage] • Give an example program whose result can differ under different rules for evaluation order, precedence, or associativity [Usage] • Show uses of delayed evaluation, such as user-defined control abstractions [Familiarity] • Discuss the need for allowing calls to external calls and system libraries and the consequences for language implementation [Familiarity]
Readings: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIT 3: Type Systems (18)**Competences:****Content**

- Compositional type constructors, such as product types (for aggregates), sum types (for unions), function types, quantified types, and recursive types
- Type checking
- Type safety as preservation plus progress
- Type inference
- Static overloading

Generales Goals

- Define a type system precisely and compositionally [Usage]
- For various foundational type constructors, identify the values they describe and the invariants they enforce [Familiarity]
- Precisely specify the invariants preserved by a sound type system [Familiarity]
- Prove type safety for a simple language in terms of preservation and progress theorems [Usage]
- Implement a unification-based type-inference algorithm for a simple language [Usage]
- Explain how static overloading and associated resolution algorithms influence the dynamic behavior of programs [Familiarity]

Readings: Sebesta (2012), Webber (2010), Roy and Haridi (2004)

UNIT 4: Object-Oriented Programming (12)**Competences:****Content****Generales Goals**

- Object-oriented design
 - Decomposition into objects carrying state and having behavior
 - Class-hierarchy design for modeling
- Definition of classes: fields, methods, and constructors
- Subclasses, inheritance, and method overriding
- Dynamic dispatch: definition of method-call
- Subtyping
 - Subtype polymorphism; implicit upcasts in typed languages
 - Notion of behavioral replacement: subtypes acting like supertypes
 - Relationship between subtyping and inheritance
- Object-oriented idioms for encapsulation
 - Privacy and visibility of class members
 - Interfaces revealing only method signatures
 - Abstract base classes
- Using collection classes, iterators, and other common library components

- Design and implement a class [Usage]
- Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses [Usage]
- Correctly reason about control flow in a program using dynamic dispatch [Usage]
- Compare and contrast (1) the procedural/functional approach—defining a function for each operation with the function body providing a case for each data variant—and (2) the object-oriented approach—defining a class for each data variant with the class definition providing a method for each operation Understand both as defining a matrix of operations and variants [Assessment]
- 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) [Usage]
- Use object-oriented encapsulation mechanisms such as interfaces and private members [Usage]
- Define and use iterators and other operations on aggregates, including operations that take functions as arguments, in multiple programming languages, selecting the most natural idioms for each language [Usage]

Readings: Sebesta (2012), Webber (2010), Roy and Haridi (2004)

UNIT 5: Functional Programming (18)	
Competences:	
Content	Generales Goals
<ul style="list-style-type: none"> • Effect-free programming <ul style="list-style-type: none"> – Function calls have no side effects, facilitating compositional reasoning – Variables are immutable, preventing unexpected changes to program data by other code – Data can be freely aliased or copied without introducing unintended effects from mutation • Processing structured data (e.g., trees) via functions with cases for each data variant <ul style="list-style-type: none"> – Associated language constructs such as discriminated unions and pattern-matching over them – Functions defined over compound data in terms of functions applied to the constituent pieces • First-class functions (taking, returning, and storing functions) • Function closures (functions using variables in the enclosing lexical environment) <ul style="list-style-type: none"> – Basic meaning and definition – creating closures at run-time by capturing the environment – Canonical idioms: call-backs, arguments to iterators, reusable code via function arguments – Using a closure to encapsulate data in its environment – Currying and partial application • Defining higher-order operations on aggregates, especially map, reduce/fold, and filter 	<ul style="list-style-type: none"> • Write basic algorithms that avoid assigning to mutable state or considering reference equality [Usage] • Write useful functions that take and return other functions [Usage] • Compare and contrast (1) the procedural/functional approach-defining a function for each operation with the function body providing a case for each data variant-and (2) the object-oriented approach-defining a class for each data variant with the class definition providing a method for each operation Understand both as defining a matrix of operations and variants [Assessment] • Correctly reason about variables and lexical scope in a program using function closures [Usage] • Use functional encapsulation mechanisms such as closures and modular interfaces [Usage] • Define and use iterators and other operations on aggregates, including operations that take functions as arguments, in multiple programming languages, selecting the most natural idioms for each language [Usage]
Readings: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIT 6: Event-Driven and Reactive Programming (12)	
Competences:	
Content	Generales Goals
<ul style="list-style-type: none"> • Events and event handlers • Canonical uses such as GUIs, mobile devices, robots, servers • Using a reactive framework <ul style="list-style-type: none"> – Defining event handlers/listeners – Main event loop not under event-handler-writer’s control • Externally-generated events and program-generated events • Separation of model, view, and controller 	<ul style="list-style-type: none"> • Write event handlers for use in reactive systems, such as GUIs [Usage] • Explain why an event-driven programming style is natural in domains where programs react to external events [Familiarity] • Describe an interactive system in terms of a model, a view, and a controller [Familiarity]
Readings: Sebesta (2012)	

UNIT 7: Logic Programming (12)	
Competences:	
Content	Generales Goals
<ul style="list-style-type: none"> • Causal representation of data structures and algorithms • Unification • Backtracking and search • Cuts 	<ul style="list-style-type: none"> • Use a logic language to implement a conventional algorithm [Usage] • Use a logic language to implement an algorithm employing implicit search using clauses, relations, and cuts [Usage]
Readings: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

8. Methodology
<p>El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.</p> <p>El profesor del curso presentará demostraciones para fundamentar clases teóricas.</p> <p>El profesor y los alumnos realizarán prácticas</p> <p>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.</p>

9. Assessment
<p>Continuous Assessment 1 : 20 %</p> <p>Partial Exam : 30 %</p> <p>Continuous Assessment 2 : 20 %</p> <p>Final exam : 30 %</p>

References

- Roy, Peter Van and Seif Haridi (2004). *Concepts, Techniques, and Models of Computer Programming*. MIT Press: Cambridge, MA, USA. ISBN: 0262220695.
- Sebesta, Robert W. (2012). *Concepts of Programming Languages*. 10th. Addison-Wesley Publishing Company: USA. ISBN: 0131395319.
- Webber, Adam Brooks (2010). *Modern Programming Languages: A Practical Introduction*. 2nd. Franklin, Beedle and Associates, Inc. ISBN: 978-1-59028-250-2.