

1. COURSE

CS341. Programming languages (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS341. Programming languages
2.2 Semester	:	7 ^{mo} Semestre.
2.3 Credits	:	4
2.4 Horas	:	3 HT; 3 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 Prerequisites	:	CS113. Programming II. (3 rd Sem) CS113. Programming II. (3 rd Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

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.

5. 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. COMPETENCES

- 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. TOPICS

Unit 1: (18)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Historia de los Lenguajes de Programación • Programas que tienen otros programas como entrada tales como interpretes, compiladores, revisores de tipos y generadores de documentación. • Estructuras de datos que representan código para ejecución, traducción o transmisión. • Estructura de un programa: Léxico, Sintáctico y Semántico • BNF • Interpretación vs. compilación a código nativo vs. compilación de representación portable intermedia. [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] • Explicar como programas que procesan otros programas tratan a los otros programas como su entrada de datos [Familiarity] • Describir un árbol de sintaxis abstracto para un lenguaje pequeño [Familiarity] • Escribir un programa para procesar alguna representación de código para algún propósito, tales como un interprete, una expresión optimizada, o un generador de documentación [Usage] • Distinguir una definición de un lenguaje de una implementación particular de un lenguaje (compilador vs interprete, tiempo de ejecución de la representación de los objetos de datos, etc) [Familiarity] • Reconocer como funciona un programa a nivel de computador. [Familiarity]
Readings : [Seb12], [Web10]	

Unit 2: Pragmática de lenguajes (12)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Principios de diseño de lenguaje tales como la ortogonalidad. • Orden de evaluación, precedencia y asociatividad. • Evaluación tardía vs. evaluación temprana. • Definiendo controles y constructos de iteración. • Llamadas externas y sistema de librerías. 	<ul style="list-style-type: none"> • Discute el rol de conceptos como ortogonalidad y el buen criterio de selección en el diseño de lenguajes [Usage] • Utiliza criterios objetivos y nítidos para evaluar las decisiones en el diseño de un lenguaje [Usage] • Da un ejemplo de un programa cuyo resultado puede diferir dado diversas reglas de orden de evaluación, precedencia, o asociatividad [Usage] • Muestra el uso de evaluación con retraso, como en el caso de abstracciones definidas y controladas por el usuario [Familiarity] • Discute la necesidad de permitir llamadas a librerías externas y del sistema y las consecuencias de su implementación en un lenguaje [Familiarity]
Readings : [Seb12], [Web10], [RH04]	

Unit 3: Sistemas de tipos (18)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Constructores de tipo composicional, como tipos de producto (para agregados), tipos de suma (para uniones), tipos de función, tipos cuantificados y tipos recursivos. • Comprobación de tipos. • Seguridad de tipos como preservación más progreso. • Inferencia de tipos. • Sobrecarga estática. 	<ul style="list-style-type: none"> • Definir un sistema de tipo de forma precisa y en su composición [Usage] • Para varias construcciones de tipo fundamental, identificar los valores que describen y las invariantes que hacen que se cumplan [Familiarity] • Precisar las invariantes preservadas por un sistema de tipos seguro (<i>sound type system</i>) [Familiarity] • Demostrar la seguridad de tipos para un lenguaje simple en términos de conservación y progreso teoremas [Usage] • Implementar un algoritmo de inferencia de tipos basado en la unificación para un lenguaje básico [Usage] • Explicar cómo la sobrecarga estática y algoritmos de resolución asociados influyen el comportamiento dinámico de los programas [Familiarity]
Readings : [Seb12], [Web10], [RH04]	

Unit 4: Programación orientada a objetos (12)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Diseño orientado a objetos: <ul style="list-style-type: none"> – Descomposición en objetos que almacenan estados y poseen comportamiento – Diseño basado en jerarquía de clases para modelamiento • Definición de las categorías, campos, métodos y constructores. • Las subclases, herencia y método de alteración temporal. • Asignación dinámica: definición de método de llamada. • Subtipificación: <ul style="list-style-type: none"> – Polimorfismo artículo Subtipo; upcasts implícitos en lenguajes con tipos. – Noción de reemplazo de comportamiento: los subtipos de actuar como supertipos. – Relación entre subtipos y la herencia. • Lenguajes orientados a objetos para la encapsulación: <ul style="list-style-type: none"> – privacidad y la visibilidad de miembros de la clase – Interfaces revelan único método de firmas – clases base abstractas • Uso de colección de clases, iteradores, y otros componentes de la librería estándar. 	<ul style="list-style-type: none"> • Diseñar e implementar una clase [Usage] • Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usage] • Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usage] • Comparar y contrastar (1) el enfoque procedur/funcional- definiendo una función por cada operación con el uso de la función proporcionando un caso por cada variación de dato - y (2) el enfoque orientado a objetos - definiendo una clase por cada variación de dato con la definición de la clase proporcionando un método por cada operación. Entender ambos enfoques como una definición de variaciones y operaciones de una matriz [Assessment] • Explicar la relación entre la herencia orientada a objetos (código compartido y <i>overriding</i>) y subtipificación (la idea de un subtipo es ser utilizable en un contexto en el que espera al supertipo) [Usage] • Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usage] • Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, seleccionar la forma más natural por cada lenguaje [Usage]
Readings : [Seb12], [Web10], [RH04]	

Unit 5: Programación funcional (18)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • El efecto de la programación libre: <ul style="list-style-type: none"> – Llamadas a función que no tiene efecto secundarios, para facilitar el razonamiento composicional – Variables inmutables, prevención de cambios no esperados en los datos del programa por otro código. – Datos que pueden ser subnombrados o copiados libremente sin introducir efectos no deseados del cambio • Procesamiento de estructuras de datos (p.e. arboles) a través de funciones con casos para cada variación de los datos. <ul style="list-style-type: none"> – Constructores asociados al lenguaje tales como uniones discriminadas y reconocimiento de patrones sobre ellos. – Funciones definidas sobre datos compuestos en términos de funciones aplicadas a las piezas constituidas. • Funciones de primera clase (obtener, retornar y funciones de almacenamiento) • Cierres de función (funciones que usan variables en entornos léxicos cerrados) <ul style="list-style-type: none"> – Significado y definición básicos - creación de cierres en tiempo de ejecución mediante la captura del entorno. – Idiomas canónicos: llamadas de retorno, argumentos de iteradores, código reusable mediante argumentos de función – Uso del cierre para encapsular datos en su entorno – Evaluación y aplicación parcial • Definición de las operaciones de orden superior en los agregados, especialmente en mapa, reducir / doblar, y el filtro. 	<ul style="list-style-type: none"> • Escribir algoritmos básicos que eviten asignación a un estado mutable o considerar igualdad de referencia [Usage] • Escribir funciones útiles que puedan tomar y retornar otras funciones [Usage] • Comparar y contrastar (1) el enfoque procedurar/funcional- definiendo una función por cada operación con el acuerdo de la función proporcionando un caso por cada variación de dato - y (2) el enfoque orientado a objetos - definiendo una clase por cada variación de dato con la definición de la clase proporcionando un método por cada operación. Entender ambos enfoques como una definición de variaciones y operaciones de una matriz [Assessment] • Razonar correctamente sobre variables y el ámbito léxico en un programa usando funciones de cierre (<i>function closures</i>) [Usage] • Usar mecanismos de encapsulamiento funcional, tal como <i>closures</i> e interfaces modulares [Usage] • Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, seleccionar la forma mas natural por cada lenguaje [Usage]
Readings : [Seb12], [Web10], [RH04]	

Unit 6: Programación reactiva y dirigida por eventos (12)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Eventos y controladores de eventos. • Usos canónicos como interfaces gráficas de usuario, dispositivos móviles, robots, servidores. • Uso de frameworks reactivos. <ul style="list-style-type: none"> – Definición de controladores/oyentes (handles/listeners) de eventos. – Bucle principal de eventos no controlado por el escritor controlador de eventos (event-handler-writer) • Eventos y eventos del programa generados externamente generada. • La separación de modelo, vista y controlador. 	<ul style="list-style-type: none"> • Escribir manejadores de eventos para su uso en sistemas reactivos tales como GUIs [Usage] • Explicar porque el estilo de programación manejada por eventos es natural en dominios donde el programa reacciona a eventos externos [Familiarity] • Describir un sistema interactivo en términos de un modelo, una vista y un controlador [Familiarity]
Readings : [Seb12]	

Unit 7: Programación lógica (12)	
Competences Expected:	
Topics	Learning Outcomes
<ul style="list-style-type: none"> • Representación causal de estructura de datos y algoritmos. • Unificación. • Backtracking y búsqueda. • Cuts. 	<ul style="list-style-type: none"> • Usa un lenguaje lógico para implementar un algoritmo convencional [Usage] • Usa un lenguaje lógico para implementar un algoritmo empleando búsqueda implícita usando cláusulas, relaciones, y cortes [Usage]
Readings : [Seb12], [Web10], [RH04]	

8. WORKPLAN

8.1 Methodology

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

8.2 Theory Sessions

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

8.3 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.

9. EVALUATION SYSTEM

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

10. BASIC BIBLIOGRAPHY

- [RH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. Cambridge, MA, USA: MIT Press, 2004. ISBN: 0262220695.
- [Seb12] Robert W. Sebesta. *Concepts of Programming Languages*. 10th. USA: Addison-Wesley Publishing Company, 2012. ISBN: 0131395319.

[Web10] Adam Brooks Webber. *Modern Programming Languages: A Practical Introduction*. 2nd. Franklin, Beedle and Associates, Inc, 2010. ISBN: 978-1-59028-250-2.