Universidad Católica San Pablo (UCSP)
Escuela Profesional de
Ciencia de la Computación
SILABO

CS341. Lenguajes de Programación (Obligatorio)

2020-I

1. Información general		
1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS341. Lenguajes de Programación
1.3 Semestre	:	7^{mo} Semestre.
1.4 Prerrequisitos	:	CS211. Teoría de la Computación. (4^{to} Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Virtual
1.7 horas	:	2 HT; 2 HP; 2 HL;
1.8 Créditos	:	4

2. Profesores

3. Fundamentación del curso

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 carácteristicas del lenguaje de programación estan 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. Resumen

1. 2. Pragmática de lenguajes 3. Sistemas de tipos 4. Programación orientada a objetos 5. Programación funcional 6. Programación reactiva y dirigida por eventos 7. Programación lógica

5. Objetivos Generales

- 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. Contribución a los resultados (*Outcomes*)

Esta disciplina contribuye al logro de los siguientes resultados de la carrera:

1) Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Usar**)

6) Aplicar fundamentos de teoria de ciencias de la computación y desarrollo de software para producir soluciones basados en computación. (**Usar**)

7. Contenido

UNIDAD 1: (18)	
Competencias:	
Contenido	**Objetivos Generales**
• 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. [Familiarizarse]	• Reconocer el desarrollo histórico de los lenguajes de programación. [Familiarizarse] • Identificar los paradigmas que agrupan a la mayoría de lenguajes de programación existentes hoy en día. [Familiarizarse] • Explicar como programas que procesan otros programas tratan a los otros programas como su entrada de datos [Familiarizarse] • Describir un árbol de sintaxis abstracto para un lenguaje pequeño [Familiarizarse] • Escribir un programa para procesar alguna representación de código para algún propósito, tales como un interprete, una expresión optimizada, o un generador de documentación [Usar] • 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) [Familiarizarse] • Reconocer como funciona un programa a nivel de computador. [Familiarizarse]
Lecturas: Sebesta (2012), Webber (2010)	

UNIDAD 2: Pragmática de lenguajes (12)	
Competencias:	
Contenido	**Objetivos Generales**
• 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.	• Discute el rol de conceptos como ortogonalidad y el buen criterio de selección en el diseño de lenguajes [Usar] • Utiliza criterios objetivos y nítidos para evaluar las decisiones en el diseño de un lenguaje [Usar] • Da un ejemplo de un programa cuyo resultado puede diferir dado diversas reglas de orden de evaluación, precedencia, o asociatividad [Usar] • Muestra el uso de evaluación con retraso, como en el caso de abstracciones definidas y controladas por el usuario [Familiarizarse] • Discute la necesidad de permitir llamadas a librerias externas y del sistema y las consecuencias de su implementación en un lenguaje [Familiarizarse]
Lecturas: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIDAD 3: Sistemas de tipos (18)	
Competencias:	
Contenido	**Objetivos Generales**
• 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.	• Definir un sistema de tipo de forma precisa y en su composición [Usar] • Para varias construcciones de tipo fundamental, identificar los valores que describen y las invariantes que hacen que se cumplan [Familiarizarse] • Precisar las invariantes preservadas por un sistema de tipos seguro (*sound type system*) [Familiarizarse] • Demostrar la seguridad de tipos para un lenguaje simple en términos de conservación y progreso teoremas [Usar] • Implementar un algoritmo de inferencia de tipos basado en la unificación para un lenguaje básico [Usar] • Explicar cómo la sobrecarga estática y algoritmos de resolución asociados influyen el comportamiento dinámico de los programas [Familiarizarse]
Lecturas: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIDAD 4: Programación orientada a objetos (12)

Competencias:

Contenido	**Objetivos Generales**
• Diseño orientado a objetos: – Descomposicion en objetos que almacenan estados y poseen comportamiento – Diseño basado en jerarquia 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: – 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: – privacidad y la visibilidad de miembros de la clase – Interfaces revelan único método de firmas – clases base abstractas • Uso de coleccion de clases, iteradores, y otros componentes de la libreria estandar.	• Diseñar e implementar una clase [Usar] • Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usar] • Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usar] • Comparar y contrastar (1) el enfoque procedurar/funcional- definiendo una función por cada operación con el cuerdo 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 [Evaluar] • Explicar la relación entre la herencia orientada a objetos (codigo compartido y *overriding*) y subtipificación (la idea de un subtipo es ser utilizable en un contexto en el que espera al supertipo) [Usar] • Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usar] • Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, selecionar la forma mas natural por cada lenguaje [Usar]

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

UNIDAD 5: Programación funcional (18)

Competencias:

Contenido	Objetivos Generales
• El efecto de la programación libre: – Llamadas a función que no tiene efecto secundarios, para facilitar el razonamiento composicional – Variables inmutables, prevencion 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 fuciones con casos para cada variación de los datos. – 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) – Significado y definicion básicos - creacion 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.	• Escribir algoritmos básicos que eviten asignación a un estado mutable o considerar igualdad de referencia [Usar] • Escribir funciones útiles que puedan tomar y retornar otras funciones [Usar] • Comparar y contrastar (1) el enfoque procedurar/funcional- definiendo una función por cada operación con el cuerdo 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 [Evaluar] • Razonar correctamente sobre variables y el ámbito léxico en un programa usando funciones de cierre (*function closures*) [Usar] • Usar mecanismos de encapsulamiento funcional, tal como *closures* e interfaces modulares [Usar] • 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 [Usar]

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

UNIDAD 6: Programación reactiva y dirigida por eventos (12)	
Competencias:	
Contenido	**Objetivos Generales**

• Eventos y controladores de eventos.
• Usos canónicos como interfaces gráficas de usuario, dispositivos móviles, robots, servidores.
• Uso de frameworks reactivos.
  – Definición de controladores/oyentes (handles/listeners) de eventos.
  – Bucle principal de enventos no controlado po el escritor controlador de eventos (event-handler-writer)
• Eventos y eventos del programa generados externamente generada.
• La separación de modelo, vista y controlador.
	• Escribir manejadores de eventos para su uso en sistemas reactivos tales como GUIs [Usar]
	• Explicar porque el estilo de programación manejada por eventos es natural en dominios donde el programa reacciona a eventos externos [Familiarizarse]
	• Describir un sistema interactivo en términos de un modelo, una vista y un controlador [Familiarizarse]

Lecturas: Sebesta (2012)	

UNIDAD 7: Programación lógica (12)	
Competencias:	
Contenido	**Objetivos Generales**
• Representación causal de estructura de datos y algoritmos. • Unificación. • Bactracking y busqueda. • Cuts.	• Usa un lenguaje lógico para implementar un algoritmo convencional [Usar] • Usa un lenguaje lógico para implementar un algoritmo empleando búsqueda implícita usando claúsulas, relaciones, y cortes [Usar]
Lecturas: Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

8. Metodología

El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.

El profesor del curso presentará demostraciones para fundamentar clases teóricas.

El profesor y los alumnos realizarán prácticas

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. Evaluar

Evaluación Continua 1 : 20 %

Examen parcial : 30 %

Evaluación Continua 2 : 20 %

Examen final : 30 %

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.