

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



**CS341. Lenguajes de Programación (Obligatorio)**

**1. Información general**

1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS341. Lenguajes de Programación
1.3 Semestre	:	7 <sup>mo</sup> Semestre.
1.4 Prerrequisitos	:	CS211. Teoría de la Computación. (4 <sup>to</sup> Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Presencial
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 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. 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:

- a) Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (**Usar**)
- b) Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución. (**Usar**)
- i) Utilizar técnicas y herramientas actuales necesarias para la práctica de la computación. (**Usar**)
- j) Aplicar la base matemática, principios de algoritmos y la teoría de la CS en el modelamiento y diseño de sistemas. (**Usar**)

**7. Contenido**

<b>UNIDAD 1: (18)</b>	
<b>Competencias: a,b,i,j</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Historia de los Lenguajes de Programación</li> <li>• Programas que tienen otros programas como entrada tales como interpretes, compiladores, revisores de tipos y generadores de documentación.</li> <li>• Estructuras de datos que representan código para ejecución, traducción o transmisión.</li> <li>• Estructura de un programa: Léxico, Sintáctico y Semántico</li> <li>• BNF</li> <li>• Interpretación vs. compilación a código nativo vs. compilación de representación portable intermedia. [Familiarizarse]</li> </ul>	<ul style="list-style-type: none"> <li>• Reconocer el desarrollo histórico de los lenguajes de programación. [Familiarizarse]</li> <li>• Identificar los paradigmas que agrupan a la mayoría de lenguajes de programación existentes hoy en día. [Familiarizarse]</li> <li>• Explicar como programas que procesan otros programas tratan a los otros programas como su entrada de datos [Familiarizarse]</li> <li>• Describir un árbol de sintaxis abstracto para un lenguaje pequeño [Familiarizarse]</li> <li>• 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]</li> <li>• 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]</li> <li>• Reconocer como funciona un programa a nivel de computador. [Familiarizarse]</li> </ul>
<b>Lecturas:</b> Sebesta (2012), Webber (2010)	

<b>UNIDAD 2: Pragmática de lenguajes (12)</b>	
<b>Competencias: a,b,i,j</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Principios de diseño de lenguaje tales como la ortogonalidad.</li> <li>• Orden de evaluación, precedencia y asociatividad.</li> <li>• Evaluación tardía vs. evaluación temprana.</li> <li>• Definiendo controles y constructos de iteración.</li> <li>• Llamadas externas y sistema de librerías.</li> </ul>	<ul style="list-style-type: none"> <li>• Discute el rol de conceptos como ortogonalidad y el buen criterio de selección en el diseño de lenguajes [Usar]</li> <li>• Utiliza criterios objetivos y nítidos para evaluar las decisiones en el diseño de un lenguaje [Usar]</li> <li>• Da un ejemplo de un programa cuyo resultado puede diferir dado diversas reglas de orden de evaluación, precedencia, o asociatividad [Usar]</li> <li>• Muestra el uso de evaluación con retraso, como en el caso de abstracciones definidas y controladas por el usuario [Familiarizarse]</li> <li>• Discute la necesidad de permitir llamadas a librerías externas y del sistema y las consecuencias de su implementación en un lenguaje [Familiarizarse]</li> </ul>
<b>Lecturas:</b> Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIDAD 3: Sistemas de tipos (18)	
Competencias: a,b,i,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Constructores de tipo composicional, como tipos de producto (para agregados), tipos de suma (para uniones), tipos de función, tipos cuantificados y tipos recursivos.</li> <li>• Comprobación de tipos.</li> <li>• Seguridad de tipos como preservación más progreso.</li> <li>• Inferencia de tipos.</li> <li>• Sobrecarga estática.</li> </ul>	<ul style="list-style-type: none"> <li>• Definir un sistema de tipo de forma precisa y en su composición [Usar]</li> <li>• Para varias construcciones de tipo fundamental, identificar los valores que describen y las invariantes que hacen que se cumplan [Familiarizarse]</li> <li>• Precisar las invariantes preservadas por un sistema de tipos seguro (<i>sound type system</i>) [Familiarizarse]</li> <li>• Demostrar la seguridad de tipos para un lenguaje simple en términos de conservación y progreso teoremas [Usar]</li> <li>• Implementar un algoritmo de inferencia de tipos basado en la unificación para un lenguaje básico [Usar]</li> <li>• Explicar cómo la sobrecarga estática y algoritmos de resolución asociados influyen el comportamiento dinámico de los programas [Familiarizarse]</li> </ul>
<b>Lecturas:</b> Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIDAD 4: Programación orientada a objetos (12)	
Competencias: a,b,i,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Diseño orientado a objetos: <ul style="list-style-type: none"> <li>– Descomposición en objetos que almacenan estados y poseen comportamiento</li> <li>– Diseño basado en jerarquía de clases para modelamiento</li> </ul> </li> <li>• Definición de las categorías, campos, métodos y constructores.</li> <li>• Las subclases, herencia y método de alteración temporal.</li> <li>• Asignación dinámica: definición de método de llamada.</li> <li>• Subtipificación: <ul style="list-style-type: none"> <li>– Polimorfismo artículo Subtipo; upcasts implícitos en lenguajes con tipos.</li> <li>– Noción de reemplazo de comportamiento: los subtipos de actuar como supertipos.</li> <li>– Relación entre subtipos y la herencia.</li> </ul> </li> <li>• Lenguajes orientados a objetos para la encapsulación: <ul style="list-style-type: none"> <li>– privacidad y la visibilidad de miembros de la clase</li> <li>– Interfaces revelan único método de firmas</li> <li>– clases base abstractas</li> </ul> </li> <li>• Uso de colección de clases, iteradores, y otros componentes de la librería estándar.</li> </ul>	<ul style="list-style-type: none"> <li>• Diseñar e implementar una clase [Usar]</li> <li>• Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usar]</li> <li>• Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usar]</li> <li>• Comparar y contrastar (1) el enfoque proceduracional/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 [Evaluar]</li> <li>• 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) [Usar]</li> <li>• Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usar]</li> <li>• 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 [Usar]</li> </ul>
<b>Lecturas:</b> Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

UNIDAD 5: Programación funcional (18)	
Competencias: a,b,i,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• El efecto de la programación libre: <ul style="list-style-type: none"> <li>– Llamadas a función que no tiene efecto secundarios, para facilitar el razonamiento composicional</li> <li>– Variables inmutables, prevención de cambios no esperados en los datos del programa por otro código.</li> <li>– Datos que pueden ser subnombrados o copiados libremente sin introducir efectos no deseados del cambio</li> </ul> </li> <li>• 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"> <li>– Constructores asociados al lenguaje tales como uniones discriminadas y reconocimiento de patrones sobre ellos.</li> <li>– Funciones definidas sobre datos compuestos en términos de funciones aplicadas a las piezas constituidas.</li> </ul> </li> <li>• Funciones de primera clase (obtener, retornar y funciones de almacenamiento)</li> <li>• Cierres de función (funciones que usan variables en entornos léxicos cerrados) <ul style="list-style-type: none"> <li>– Significado y definición básicos - creación de cierres en tiempo de ejecución mediante la captura del entorno.</li> <li>– Idiomas canónicos: llamadas de retorno, argumentos de iteradores, código reusable mediante argumentos de función</li> <li>– Uso del cierre para encapsular datos en su entorno</li> <li>– Evaluación y aplicación parcial</li> </ul> </li> <li>• Definición de las operaciones de orden superior en los agregados, especialmente en mapa, reducir / doblar, y el filtro.</li> </ul>	<ul style="list-style-type: none"> <li>• Escribir algoritmos básicos que eviten asignación a un estado mutable o considerar igualdad de referencia [Usar]</li> <li>• Escribir funciones útiles que puedan tomar y retornar otras funciones [Usar]</li> <li>• Comparar y contrastar (1) el enfoque procedurador/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]</li> <li>• Razonar correctamente sobre variables y el ámbito léxico en un programa usando funciones de cierre (<i>function closures</i>) [Usar]</li> <li>• Usar mecanismos de encapsulamiento funcional, tal como <i>closures</i> e interfaces modulares [Usar]</li> <li>• 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]</li> </ul>
<b>Lecturas:</b> Sebesta (2012), Webber (2010), Roy and Haridi (2004)	

<b>UNIDAD 6: Programación reactiva y dirigida por eventos (12)</b>	
<b>Competencias: a,b,i,j</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Eventos y controladores de eventos.</li> <li>• Usos canónicos como interfaces gráficas de usuario, dispositivos móviles, robots, servidores.</li> <li>• Uso de frameworks reactivos. <ul style="list-style-type: none"> <li>– Definición de controladores/oyentes (handles/listeners) de eventos.</li> <li>– Bucle principal de eventos no controlado por el escritor controlador de eventos (event-handler-writer)</li> </ul> </li> <li>• Eventos y eventos del programa generados externamente generada.</li> <li>• La separación de modelo, vista y controlador.</li> </ul>	<ul style="list-style-type: none"> <li>• Escribir manejadores de eventos para su uso en sistemas reactivos tales como GUIs [Usar]</li> <li>• Explicar porque el estilo de programación manejada por eventos es natural en dominios donde el programa reacciona a eventos externos [Familiarizarse]</li> <li>• Describir un sistema interactivo en términos de un modelo, una vista y un controlador [Familiarizarse]</li> </ul>
<b>Lecturas:</b> Sebasta (2012)	

<b>UNIDAD 7: Programación lógica (12)</b>	
<b>Competencias: a,b,i,j</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Representación causal de estructura de datos y algoritmos.</li> <li>• Unificación.</li> <li>• Backtracking y búsqueda.</li> <li>• Cuts.</li> </ul>	<ul style="list-style-type: none"> <li>• Usa un lenguaje lógico para implementar un algoritmo convencional [Usar]</li> <li>• Usa un lenguaje lógico para implementar un algoritmo empleando búsqueda implícita usando cláusulas, relaciones, y cortes [Usar]</li> </ul>
<b>Lecturas:</b> Sebasta (2012), Webber (2010), Roy and Haridi (2004)	

8. Metodología
<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. Evaluar
<p><b>Evaluación Continua 1 : 20 %</b></p> <p><b>Examen parcial : 30 %</b></p> <p><b>Evaluación Continua 2 : 20 %</b></p> <p><b>Examen final : 30 %</b></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.