

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



**CS112. Ciencia de la Computación I (Obligatorio)**

**1. Información general**

1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS112. Ciencia de la Computación I
1.3 Semestre	:	2 <sup>do</sup> Semestre.
1.4 Prerrequisitos	:	CS111. Programación de Video Juegos. (1 <sup>er</sup> Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Virtual
1.7 horas	:	2 HT; 2 HP; 4 HL;
1.8 Créditos	:	5

**2. Profesores**

**3. Fundamentación del curso**

Este es el segundo curso en la secuencia de los cursos introductorios a la Ciencia de la Computación. El curso introducirá a los participantes en los diversos temas del área de computación como: algoritmos, estructuras de datos, ingeniería del software, etc.

**4. Resumen**

1. Visión General de los Lenguajes de Programación 2. Sistemas de tipos básicos 3. Conceptos Fundamentales de Programación 4. Programación orientada a objetos 5. Algoritmos y Diseño 6. Estrategias Algorítmicas 7. Análisis Básico 8. Algoritmos y Estructuras de Datos fundamentales

**5. Objetivos Generales**

- Introducir al alumno a los fundamentos del paradigma de orientación a objetos, permitiendo asimilar los conceptos necesarios para desarrollar sistemas de información.

**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. (**Evaluar**)
- c) Diseñar, implementar y evaluar un sistema, proceso, componente o programa computacional para alcanzar las necesidades deseadas. (**Evaluar**)
- h) Incorporarse a un proceso de aprendizaje profesional continuo. (**Familiarizarse**)
- i) Utilizar técnicas y herramientas actuales necesarias para la práctica de la computación. (**Usar**)

**7. Contenido**

<b>UNIDAD 1: Visión General de los Lenguajes de Programación (1)</b>	
<b>Competencias: a</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Breve revisión de los paradigmas de programación.</li> <li>• Comparación entre programación funcional y programación imperativa.</li> <li>• Historia de los lenguajes de programación.</li> </ul>	<ul style="list-style-type: none"> <li>• Discutir el contexto histórico de los paradigmas de diversos lenguajes de programación [Familiarizarse]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

UNIDAD 2: Sistemas de tipos básicos (2)	
Competencias: a,i	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Tipos como conjunto de valores junto con un conjunto de operaciones. <ul style="list-style-type: none"> <li>– Tipos primitivos (p.e. números, booleanos)</li> <li>– Composición de tipos contruídos de otros tipos (p.e., registros, uniones, arreglos, listas, funciones, referencias)</li> </ul> </li> <li>• Declaración de modelos (enlace, visibilidad, alcance y tiempo de vida).</li> <li>• Vista general del chequeo de tipos.</li> </ul>	<ul style="list-style-type: none"> <li>• Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Familiarizarse]</li> <li>• Para un lenguaje con sistema de tipos estático, describir las operaciones que están prohibidas de forma estática, como pasar el tipo incorrecto de valor a una función o método [Familiarizarse]</li> <li>• Describir ejemplos de errores de programa detectadas por un sistema de tipos [Familiarizarse]</li> <li>• Para múltiples lenguajes de programación, identificar propiedades de un programa con verificación estática y propiedades de un programa con verificación dinámica [Usar]</li> <li>• Dar un ejemplo de un programa que no verifique tipos en un lenguaje particular y sin embargo no tenga error cuando es ejecutado [Familiarizarse]</li> <li>• Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar]</li> <li>• Explicar como las reglas de tipificación definen el conjunto de operaciones que legales para un tipo [Familiarizarse]</li> <li>• Escribir las reglas de tipo que rigen el uso de un particular tipo compuesto [Usar]</li> <li>• Explicar por qué indecidibilidad requiere sistemas de tipo para conservadoramente aproximar el comportamiento de un programa [Familiarizarse]</li> <li>• Definir y usar piezas de programas (tales como, funciones, clases, métodos) que usan tipos genéricos, incluyendo para colecciones [Usar]</li> <li>• Discutir las diferencias entre, genéricos (<i>generics</i>), subtipo y sobrecarga [Familiarizarse]</li> <li>• Explicar múltiples beneficios y limitaciones de tipificación estática en escritura, mantenimiento y depuración de un software [Familiarizarse]</li> </ul>
Lecturas: Stroustrup2013, Deitel17	

<b>UNIDAD 3: Conceptos Fundamentales de Programación (6)</b>	
<b>Competencias: a,i</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Sintaxis y semántica básica de un lenguaje de alto nivel.</li> <li>• Variables y tipos de datos primitivos (ej., numeros, caracteres, booleanos)</li> <li>• Expresiones y asignaciones.</li> <li>• Operaciones básicas I/O incluyendo archivos I/O.</li> <li>• Estructuras de control condicional e iterativas.</li> <li>• Paso de funciones y parámetros.</li> </ul>	<ul style="list-style-type: none"> <li>• Analiza y explica el comportamiento de programas simples que involucran estructuras fundamentales de programación variables, expresiones, asignaciones, E/S, estructuras de control, funciones, paso de parámetros, y recursividad [Evaluar]</li> <li>• Identifica y describe el uso de tipos de datos primitivos [Familiarizarse]</li> <li>• Escribe programas que usan tipos de datos primitivos [Usar]</li> <li>• Modifica y expande programas cortos que usen estructuras de control condicionales e iterativas así como funciones [Usar]</li> <li>• Diseña, implementa, prueba, y depura un programa que usa cada una de las siguientes estructuras de datos fundamentales: cálculos básicos, E/S simple, condicional estándar y estructuras iterativas, definición de funciones, y paso de parámetros [Usar]</li> <li>• Escribe un programa que usa E/S de archivos para brindar persistencia a través de ejecuciones múltiples [Usar]</li> <li>• Escoje estructuras de condición y repetición adecuadas para una tarea de programación dada [Evaluar]</li> <li>• Describe el concepto de recursividad y da ejemplos de su uso [Familiarizarse]</li> <li>• Identifica el caso base y el caso general de un problema basado en recursividad [Evaluar]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

UNIDAD 4: Programación orientada a objetos (10)	
Competencias: a,i	
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>• 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>• Definición de las categorías, campos, métodos y constructores.</li> <li>• Las subclases, herencia y método de alteración temporal.</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>• Uso de colección de clases, iteradores, y otros componentes de la librería estándar.</li> <li>• Asignación dinámica: definición de método de llamada.</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) [Familiarizarse]</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: Stroustrup2013, Deitel17</b>	

<b>UNIDAD 5: Algoritmos y Diseño (3)</b>	
<b>Competencias: a,i</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Estrategias de solución de problemas <ul style="list-style-type: none"> <li>– Funciones matemáticas iterativas y recursivas</li> <li>– Recorrido iterativo y recursivo en estructura de datos</li> <li>– Estrategias Divide y Conquistar</li> </ul> </li> <li>• Rol de los algoritmos en el proceso de solución de problemas</li> <li>• Estrategias de solución de problemas <ul style="list-style-type: none"> <li>– Funciones matemáticas iterativas y recursivas</li> <li>– Recorrido iterativo y recursivo en estructura de datos</li> <li>– Estrategias Divide y Conquistar</li> </ul> </li> <li>• Conceptos y principios fundamentales de diseño <ul style="list-style-type: none"> <li>– Abstracción</li> <li>– Descomposición de Program</li> <li>– Encapsulamiento y camuflaje de información</li> <li>– Separación de comportamiento y aplicación</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Discute la importancia de los algoritmos en el proceso de solución de un problema [Familiarizarse]</li> <li>• Discute como un problema puede ser resuelto por múltiples algoritmos, cada uno con propiedades diferentes [Familiarizarse]</li> <li>• Crea algoritmos para resolver problemas simples [Usar]</li> <li>• Usa un lenguaje de programación para implementar, probar, y depurar algoritmos para resolver problemas simples [Usar]</li> <li>• Implementa, prueba, y depura funciones recursivas simples y sus procedimientos [Usar]</li> <li>• Determina si una solución iterativa o recursiva es la más apropiada para un problema [Evaluar]</li> <li>• Implementa un algoritmo de divide y vencerás para resolver un problema [Usar]</li> <li>• Aplica técnicas de descomposición para dividir un programa en partes más pequeñas [Usar]</li> <li>• Identifica los componentes de datos y el comportamiento de múltiples tipos de datos abstractos [Usar]</li> <li>• Implementa un tipo de dato abstracto coherente, con la menor pérdida de acoplamiento entre componentes y comportamientos [Usar]</li> <li>• Identifica las fortalezas y las debilidades relativas entre múltiples diseños e implementaciones de un problema [Evaluar]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

<b>UNIDAD 6: Estrategias Algorítmicas (3)</b>	
<b>Competencias: a,i</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Algoritmos de fuerza bruta.</li> <li>• Algoritmos voraces.</li> <li>• Divide y vencerás.</li> <li>• Backtracking recursivo.</li> <li>• Programación Dinámica.</li> </ul>	<ul style="list-style-type: none"> <li>• Para cada una de las estrategias (fuerza bruta, algoritmo goloso, divide y vencerás, recursividad en reversa y programación dinámica), identifica un ejemplo práctico en el cual se pueda aplicar [Familiarizarse]</li> <li>• Utiliza un enfoque voraz para resolver un problema específico y determina si la regla escogida lo guía a una solución óptima [Evaluar]</li> <li>• Usa un algoritmo de divide-y-vencerás para resolver un determinado problema [Usar]</li> <li>• Usa recursividad en reversa a fin de resolver un problema como en el caso de recorrer un laberinto [Usar]</li> <li>• Usa programación dinámica para resolver un problema determinado [Usar]</li> <li>• Determina el enfoque algorítmico adecuado para un problema [Evaluar]</li> <li>• Describe varios métodos basados en heurísticas para resolver problemas [Familiarizarse]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

<b>UNIDAD 7: Análisis Básico (2)</b>	
<b>Competencias: a,i</b>	
<b>Contenido</b>	<b>Objetivos Generales</b>
<ul style="list-style-type: none"> <li>• Diferencias entre el mejor, el esperado y el peor caso de un algoritmo.</li> </ul>	<ul style="list-style-type: none"> <li>• Explique a que se refiere con “mejor”, “esperado” y “peor” caso de comportamiento de un algoritmo [Familiarizarse]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

UNIDAD 8: Algoritmos y Estructuras de Datos fundamentales (6)	
Competencias: a,i	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Algoritmos numéricos simples, tales como el cálculo de la media de una lista de números, encontrar el mínimo y máximo.</li> <li>• Algoritmos de búsqueda secuencial y binaria.</li> <li>• Algoritmos de ordenamiento de peor caso cuadrático (selección, inserción)</li> <li>• Algoritmos de ordenamiento con peor caso o caso promedio en <math>O(N \lg N)</math> (Quicksort, Heapsort, Mergesort)</li> </ul>	<ul style="list-style-type: none"> <li>• Implementar algoritmos numéricos básicos [Usar]</li> <li>• Implementar algoritmos de búsqueda simple y explicar las diferencias en sus tiempos de complejidad [Evaluar]</li> <li>• Ser capaz de implementar algoritmos de ordenamiento comunes cuadráticos y <math>O(N \log N)</math> [Usar]</li> <li>• Discutir el tiempo de ejecución y eficiencia de memoria de los principales algoritmos de ordenamiento, búsqueda y hashing [Familiarizarse]</li> <li>• Discutir factores otros que no sean eficiencia computacional que influyan en la elección de algoritmos, tales como tiempo de programación, mantenibilidad, y el uso de patrones específicos de la aplicación en los datos de entrada [Familiarizarse]</li> <li>• Explicar como el balanceamiento del arbol afecta la eficiencia de varias operaciones de un arbol de búsqueda binaria [Familiarizarse]</li> <li>• Demostrar habilidad para evaluar algoritmos, para seleccionar de un rango de posibles opciones, para proveer una justificación por esa selección, y para implementar el algoritmo en un contexto en específico [Evaluar]</li> <li>• Trazar y/o implementar un algoritmo de comparación de string [Usar]</li> </ul>
<b>Lecturas: Stroustrup2013, Deitel17</b>	

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>