

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 ^{do} Semestre.
1.4 Prerrequisitos	:	CS111. Programación de Video Juegos. (1 ^{er} Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Presencial
1.7 horas	:	2 HT; 6 HP;
1.8 Créditos	:	5
1.9 Plan	:	Plan Curricular 2016

2. Profesores

Titular

- Alvaro Henry Mamani-Aliaga <ahmamani@ucsp.edu.pe>
 - Doctor en Ciencia de la Computación, UNSA, Perú, 2019.
 - Master en Ciencia de la Computación, IME-USP, Brasil, 2011.
- Manuel Loaiza Fernandez <meloaza@ucsp.edu.pe>
 - Doctor en Informatica, Pontificia Universidad Católica do Rio de Janeiro (PUC-RIO), Brasil, 2009.
 - Master en Informatica, Pontificia Universidad Católica do Rio de Janeiro (PUC-RIO), Brasil, 2005.

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. Máquinas virtuales 3. Sistemas de tipos básicos 4. Conceptos Fundamentales de Programación 5. Funciones 6. Arreglos y Punteros 7. Programación orientada a objetos 8. Plantillas y STL 9. Conceptos Avanzados

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:

- 1) S.O. Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Evaluar**)
- 2) S.O. Diseñar, implementar y evaluar una solución basada en computación para cumplir con un conjunto determinado de requisitos computacionales en el contexto de las disciplinas del programa. (**Evaluar**)
- 5) S.O. Funcionar efectivamente como miembro o líder de un equipo involucrado en actividades apropiadas a la disciplina del programa. (**Familiarizarse**)
- 6) S.O. Aplicar la teoría de la computación y los fundamentos del desarrollo de software para producir soluciones basadas en computación. . (**Usar**)

7. Contenido**UNIDAD 1: Visión General de los Lenguajes de Programación (1)****Competencias: 1**

Contenido	Objetivos Generales
<ul style="list-style-type: none"> • Breve revisión de los paradigmas de programación. • Comparación entre programación funcional y programación imperativa. • Historia de los lenguajes de programación. 	<ul style="list-style-type: none"> • Discutir el contexto histórico de los paradigmas de diversos lenguajes de programación [Familiarizarse]

Lecturas: Stroustrup2013, Deitel17**UNIDAD 2: Máquinas virtuales (2)****Competencias: 1,6**

Contenido	Objetivos Generales
<ul style="list-style-type: none"> • El concepto de máquina virtual. • Tipos de virtualización (incluyendo Hardware / Software, OS, Servidor, Servicio, Red) . • Lenguajes intermedios. 	<ul style="list-style-type: none"> • Explicar el concepto de memoria virtual y la forma cómo se realiza en hardware y software [Familiarizarse] • Diferenciar emulación y el aislamiento [Familiarizarse] • Evaluar virtualización de compensaciones [Evaluar]

Lecturas: Stroustrup2013, Deitel17

UNIDAD 3: Sistemas de tipos básicos (6)	
Competencias: 1,6	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> • Tipos como conjunto de valores junto con un conjunto de operaciones. <ul style="list-style-type: none"> – Tipos primitivos (p.e. números, booleanos) – Composición de tipos contruídos de otros tipos (p.e., registros, uniones, arreglos, listas, funciones, referencias) • Declaración de modelos (enlace, visibilidad, alcance y tiempo de vida). • Vista general del chequeo de tipos. 	<ul style="list-style-type: none"> • Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Familiarizarse] • 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] • Describir ejemplos de errores de programa detectadas por un sistema de tipos [Familiarizarse] • Para múltiples lenguajes de programación, identificar propiedades de un programa con verificación estática y propiedades de un programa con verificación dinámica [Usar] • Dar un ejemplo de un programa que no verifique tipos en un lenguaje particular y sin embargo no tenga error cuando es ejecutado [Familiarizarse] • Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar] • Explicar como las reglas de tipificación definen el conjunto de operaciones que legales para un tipo [Familiarizarse] • Escribir las reglas de tipo que rigen el uso de un particular tipo compuesto [Usar] • Explicar por qué indecidibilidad requiere sistemas de tipo para conservadoramente aproximar el comportamiento de un programa [Familiarizarse] • Definir y usar piezas de programas (tales como, funciones, clases, métodos) que usan tipos genéricos, incluyendo para colecciones [Usar] • Discutir las diferencias entre, genéricos (<i>generics</i>), subtipo y sobrecarga [Familiarizarse] • Explicar múltiples beneficios y limitaciones de tipificación estática en escritura, mantenimiento y depuración de un software [Familiarizarse]
Lecturas: Stroustrup2013, Deitel17	

UNIDAD 4: Conceptos Fundamentales de Programación (10)	
Competencias: 1,6	
Contenido	Objetivos Generales
<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 ● Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Familiarizarse] ● Sintaxis y semántica básica de un lenguaje de alto nivel. ● Variables y tipos de datos primitivos (ej., números, caracteres, booleanos) ● Expresiones y asignaciones. ● Estructuras de control condicional e iterativas. 	<ul style="list-style-type: none"> ● 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] ● Identifica y describe el uso de tipos de datos primitivos [Familiarizarse] ● Escribe programas que usan tipos de datos primitivos [Usar] ● Modifica y expande programas cortos que usen estructuras de control condicionales e iterativas así como funciones [Usar] ● 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] ● Elige estructuras de condición y repetición adecuadas para una tarea de programación dada [Evaluar]
Lecturas: Stroustrup2013, Deitel17	

UNIDAD 5: Funciones (3)	
Competencias: 1,6	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> ● Paso de funciones y parámetros. ● Paso de parámetros ● Sobrecarga en funciones ● Fundamentos de la recursividad ● Conceptos de plantillas en funciones 	<ul style="list-style-type: none"> ● 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] ● Entiende y aplica el concepto de paso de parámetros a una función, tanto por valor como por referencia.[Usar] ● Identifica y aplica el concepto de sobrecarga de funciones.[Usar] ● Describe el concepto de recursividad y da ejemplos de su uso [Familiarizarse] ● Diseña, implementa y aplica el concepto de plantillas asociado a la necesidad de crear funciones genéricas.[Usar]
Lecturas: Stroustrup2013, Deitel17	

UNIDAD 6: Arreglos y Punteros (3)	
Competencias: 1,6	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> • Definición de arreglos • Arreglos multidimensionales • Fundamentos sobre punteros • Administración dinámica de memoria • Conceptos avanzados de Punteros 	<ul style="list-style-type: none"> • Entiende e implementa arreglos unidimensionales. [Familiarizarse] • Diseña y aplica el concepto de arreglos multidimensionales. [Usar] • Entiende y aplica el concepto de referencias y punteros. [Familiarizarse] • Entiende, aplica y evalúa la relación entre punteros y arreglos. [Evaluar] • Entiende e implementa la gestión dinámica de la memoria. Diferenciando las regiones de memoria: heap y stack. [Evaluar] • Diseña, implementa y evalúa el concepto de puntero a puntero, puntero a función, entre otros conceptos. [Evaluar]
Lecturas: Stroustrup2013, Deitel17	

UNIDAD 7: Programación orientada a objetos (2)	
Competencias: 1,6	
Contenido	Objetivos Generales
<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 ● 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 ● Definición de las categorías, campos, métodos y constructores. ● Las subclases, herencia y método de alteración temporal. ● 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. ● Uso de colección de clases, iteradores, y otros componentes de la librería estándar. ● Asignación dinámica: definición de método de llamada. 	<ul style="list-style-type: none"> ● 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 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] ● 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] ● 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, seleccionar la forma más natural por cada lenguaje [Usar]
Lecturas: Stroustrup2013, Deitel17	

UNIDAD 8: Plantillas y STL (2)	
Competencias: 1,6	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> ● Definición de plantillas en clases ● Conceptos básicos sobre la Standard Template Library (STL) 	<ul style="list-style-type: none"> ● Entiende los conceptos de plantillas en clases. [Familiarizarse] ● Implementa y crea nuevos tipos de datos genéricos. [Usar] ● Entiende las estructuras básicas de la STL. [Familiarizarse] ● Usa las estructuras de datos básicas como: pila, cola, lista, vector contenidos en la STL. [Usar]
Lecturas: Stroustrup2013, Deitel2017	

UNIDAD 9: Conceptos Avanzados (2)	
Competencias: 1,6	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> Definición de sobrecarga de operadores Manipulación de entrada y salida de datos (I/O) Patrones de diseño 	<ul style="list-style-type: none"> Entiende los conceptos de sobrecarga de operadores. [Familiarizarse] Implementa la sobrecarga de operadores permitidos en el lenguaje de programación. [Usar] Entiende los conceptos de manipulación de archivos. [Familiarizarse] Crea programas de lectura y escrita en archivos. [Usar] Entiende los conceptos de patrones de diseño. [Familiarizarse]
Lecturas: Stroustrup2013, Deitel2017	

8. Metodología

1. El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.
2. El profesor del curso presentará demostraciones para fundamentar clases teóricas.
3. El profesor y los alumnos realizarán prácticas
4. 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 Sesiones Teóricas:

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

Sesiones Prácticas:

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

Sistema de Evaluación:

La nota final se obtiene a través de:

EVALUACIONES PERMANENTES	EVALUACIONES
Evaluación Permanente 1 : 20 %	Evaluación Parcial : 20 %
Evaluación Permanente 2 : 40 %	Evaluación Final : 20 %
60%	40%

Donde:

Evaluación Permanente: Comprende trabajos grupales, participación activa en clase, test de ejercicios.

- Permanente 1 (Semanas 1 - 9)
- Permanente 2 (Semanas 10 - 17)

Para aprobar el curso, el alumno debe obtener 11.5 o más en la nota final.