

Contents

Contents	1
1. Curso	2
2. Información general	2
3. Profesores	2
4. Introducción al curso	2
5. Objetivos	2
6. Competencias	2
7. Temas	2
8. Plan de trabajo	8
8.1 Metodología	8
8.2 Sesiones Teóricas	8
8.3 Sesiones Prácticas	8
9. Planificación	8
10. Sistema de Evaluación	8
11. Bibliografía básica	9

Universidad de Piura (UDEP)
Sílabo 2022-I

1. CURSO

CS112. Ciencia de la Computación I (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Créditos	: 5
2.2 Horas de teoría	: 2 (Semanal)
2.3 Horas de práctica	: 4 (Semanal)
2.4 Duración del periodo	: 16 semanas
2.5 Condición	: Obligatorio
2.6 Modalidad	: Presencial
2.7 Prerrequisitos	: CS111. Introducción a la Ciencia de la Computación. (1 ^{er} Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL 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.

5. OBJETIVOS

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

6. COMPETENCIAS

Nooutcomes

Nospecificoutcomes

7. TEMAS

Unidad 1: Visión General de los Lenguajes de Programación (1)	
Competencias esperadas: a	
Temas	Objetivos de Aprendizaje
<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 : [Str13], [Dei17]	

Unidad 2: Sistemas de tipos básicos (2)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<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 : [Str13], [Dei17]	

Unidad 3: Conceptos Fundamentales de Programación (6)**Competencias esperadas: a,b,i**

Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none">• Sintaxis y semántica básica de un lenguaje de alto nivel.• Variables y tipos de datos primitivos (ej., números, caracteres, booleanos)• Expresiones y asignaciones.• Operaciones básicas I/O incluyendo archivos I/O.• Estructuras de control condicional e iterativas.• Paso de funciones y parámetros.	<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]• Escribe un programa que usa E/S de archivos para brindar persistencia a través de ejecuciones múltiples [Usar]• Escoje estructuras de condición y repetición adecuadas para una tarea de programación dada [Evaluar]• Describe el concepto de recursividad y da ejemplos de su uso [Familiarizarse]• Identifica el caso base y el caso general de un problema basado en recursividad [Evaluar]
Lecturas : [Str13], [Dei17]	

Unidad 4: Programación orientada a objetos (10)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<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 subclasses, 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 subclasses [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 : [Str13], [Dei17]	

Unidad 5: Algoritmos y Diseño (3)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> • Estrategias de solución de problemas <ul style="list-style-type: none"> – Funciones matemáticas iterativas y recursivas – Recorrido iterativo y recursivo en estructura de datos – Estrategias Divide y Conquistar • Rol de los algoritmos en el proceso de solución de problemas • Estrategias de solución de problemas <ul style="list-style-type: none"> – Funciones matemáticas iterativas y recursivas – Recorrido iterativo y recursivo en estructura de datos – Estrategias Divide y Conquistar • Conceptos y principios fundamentales de diseño <ul style="list-style-type: none"> – Abstracción – Descomposición de Program – Encapsulamiento y camuflaje de información – Separación de comportamiento y aplicación 	<ul style="list-style-type: none"> • Discute la importancia de los algoritmos en el proceso de solución de un problema [Familiarizarse] • Discute como un problema puede ser resuelto por múltiples algoritmos, cada uno con propiedades diferentes [Familiarizarse] • Crea algoritmos para resolver problemas simples [Usar] • Usa un lenguaje de programación para implementar, probar, y depurar algoritmos para resolver problemas simples [Usar] • Implementa, prueba, y depura funciones recursivas simples y sus procedimientos [Usar] • Determina si una solución iterativa o recursiva es la más apropiada para un problema [Evaluar] • Implementa un algoritmo de divide y vencerás para resolver un problema [Usar] • Aplica técnicas de descomposición para dividir un programa en partes más pequeñas [Usar] • Identifica los componentes de datos y el comportamiento de múltiples tipos de datos abstractos [Usar] • Implementa un tipo de dato abstracto coherente, con la menor pérdida de acoplamiento entre componentes y comportamientos [Usar] • Identifica las fortalezas y las debilidades relativas entre múltiples diseños e implementaciones de un problema [Evaluar]
Lecturas : [Str13], [Dei17]	

Unidad 6: Estrategias Algorítmicas (3)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> • Algoritmos de fuerza bruta. • Algoritmos voraces. • Divide y vencerás. • Backtracking recursivo. • Programación Dinámica. 	<ul style="list-style-type: none"> • 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] • Utiliza un enfoque voraz para resolver un problema específico y determina si la regla escogida lo guía a una solución óptima [Evaluar] • Usa un algoritmo de divide-y-vencerás para resolver un determinado problema [Usar] • Usa recursividad en reversa a fin de resolver un problema como en el caso de recorrer un laberinto [Usar] • Usa programación dinámica para resolver un problema determinado [Usar] • Determina el enfoque algorítmico adecuado para un problema [Evaluar] • Describe varios métodos basados en heurísticas para resolver problemas [Familiarizarse]
Lecturas : [Str13], [Dei17]	

Unidad 7: Análisis Básico (2)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> • Diferencias entre el mejor, el esperado y el peor caso de un algoritmo. 	<ul style="list-style-type: none"> • Explique a que se refiere con “mejor”, “esperado” y “peor” caso de comportamiento de un algoritmo [Familiarizarse]
Lecturas : [Str13], [Dei17]	

Unidad 8: Algoritmos y Estructuras de Datos fundamentales (6)	
Competencias esperadas: a,b,i	
Temas	Objetivos de Aprendizaje
<ul style="list-style-type: none"> • Algoritmos numéricos simples, tales como el cálculo de la media de una lista de números, encontrar el mínimo y máximo. • Algoritmos de búsqueda secuencial y binaria. • Algoritmos de ordenamiento de peor caso cuadrático (selección, inserción) • Algoritmos de ordenamiento con peor caso o caso promedio en $O(N \lg N)$ (Quicksort, Heapsort, Mergesort) 	<ul style="list-style-type: none"> • Implementar algoritmos numéricos básicos [Usar] • Implementar algoritmos de búsqueda simple y explicar las diferencias en sus tiempos de complejidad [Evaluar] • Ser capaz de implementar algoritmos de ordenamiento comunes cuadráticos y $O(N \log N)$ [Usar] • Discutir el tiempo de ejecución y eficiencia de memoria de los principales algoritmos de ordenamiento, búsqueda y hashing [Familiarizarse] • 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] • Explicar como el balanceamiento del árbol afecta la eficiencia de varias operaciones de un árbol de búsqueda binaria [Familiarizarse] • 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 específico [Evaluar] • Trazar y/o implementar un algoritmo de comparación de string [Usar]
Lecturas : [Str13], [Dei17]	

8. PLAN DE TRABAJO

8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

8.2 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.

8.3 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.

9. PLANIFICACIÓN

FECHA	HORARIO	TIPO DE SESIÓN	DOCENTE
Consultar en EDU	Consultar en EDU	Consultar en EDU	Consultar en EDU

10. SISTEMA DE EVALUACIÓN

11. BIBLIOGRAFÍA BÁSICA

- [Dei17] Deitel & Deitel. *C++17 - The Complete Guide*. 10th. Pearson, 2017. ISBN: 978-0201734843.
- [Str13] Bjarne Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley, 2013. ISBN: 978-0-321-56384-2.