



Universidad Nacional de Ingeniería (UNI)

Escuela Profesional de
Ciencia de la Computación
Sílabo 2026-I

1. CURSO

CS111. Introducción a la Programación (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	: CS111. Introducción a la Programación
2.2 Semestre	: 1 ^{er} Semestre.
2.3 Créditos	: 4
2.4 Horas	: 2 HT; 4 HP;
2.5 Duración del periodo	: 16 semanas
2.6 Condición	: Obligatorio
2.7 Modalidad de aprendizaje	: Presencial
2.8 Prerrequisitos	: Ninguno

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

Este es el primer curso de la secuencia de cursos introductorios a la Ciencia de la Computación. Este curso introduce a los participantes a los conceptos fundamentales de programación usando Python. Los temas incluyen tipos de datos, estructuras de control, funciones, listas, recursión, y la mecánica de ejecución, prueba y depuración. La programación es uno de los pilares de la Ciencia de la Computación; cualquier profesional en el área necesitará programar para materializar sus modelos y propuestas.

5. OBJETIVOS

- Introducir los conceptos fundamentales de programación.
- Desarrollar la capacidad de abstracción usando lenguaje de programación.
- Desarrollar habilidades de resolución de problemas mediante el pensamiento algorítmico.
- Comprender estructuras de datos básicas y sus aplicaciones.
- Dominar el lenguaje de programación Python para la resolución de problemas.

6. RESULTADOS DEL ESTUDIANTE

AG-C08) Análisis de Problemas: Identifica, formula y analiza problemas complejos de computación. (Familiarity)

AG-C12) Usa enfoques sistémicos para seleccionar, desarrollar, aplica, integrar y administrar tecnologías seguras de computación para lograr los objetivos del usuario. (Usage)

7. TEMAS

Unidad 1: Introducción a la Programación y su Historia (4 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
- Breve historia de la computación y los lenguajes de programación. - Visión general del lenguaje de programación Python. - Entornos y herramientas de programación. - Estructura básica y ejecución de programas. - Escritura, ejecución y depuración de programas simples.	- Identificar tendencias importantes en la historia de la computación. [Familiarizarse] - Discutir el contexto histórico de los paradigmas de lenguajes de programación. [Familiarizarse] - Comparar la vida diaria antes y después de las computadoras personales e Internet. [Evaluar] - Escribir, compilar y ejecutar un programa simple en Python. [Usar]
Lecturas : [Zel10], [Gut13], [BB19]	

Unidad 2: Sistemas de Tipos I: Fundamentos (6 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
- Un tipo como un conjunto de valores junto con un conjunto de operaciones: - Tipos primitivos (por ejemplo, números, booleanos) - Tipos compuestos contruidos a partir de otros tipos (por ejemplo, registros/estructuras, uniones, arreglos, listas, funciones, referencias usando operaciones de conjuntos) - Seguridad de tipos y errores causados por usar valores inconsistentemente dados sus tipos pretendidos - Objetivos y limitaciones del tipado estático y dinámico: detectar y eliminar errores lo antes posible. - Tipos de datos primitivos en Python (int, float, str, bool). - Conversión de tipos y casting.	- Describir, tanto para un tipo primitivo como para uno compuesto, los valores que tienen ese tipo [Familiarizarse] - Describir ejemplos de errores de programa detectados por un sistema de tipos [Familiarizarse] - Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar] - Explicar el sistema de tipado dinámico de Python. [Familiarizarse]
Lecturas : [Zel10], [Gut13]	

Unidad 3: Programación Orientada a Objetos I: Fundamentos (14 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Programación imperativa como subconjunto de la programación orientada a objetos. • Definición de clases: campos, métodos y constructores. • Propiedades dinámicas vs estáticas. • Variables, expresiones y asignaciones. • Operaciones de entrada y salida. • Sentencias condicionales (if, elif, else). • Estructuras de iteración (bucles for, bucles while). • Bucles anidados y sentencias de control de bucle. • Fundamentos del manejo de excepciones.	• Componer una clase a través del diseño, implementación y prueba para cumplir con los requisitos de comportamiento [Usar] • Predecir y validar el flujo de control en un programa usando despacho dinámico [Evaluar] • Usar mecanismos de encapsulación orientados a objetos como interfaces y miembros privados [Usar] • Implementar programas usando estructuras condicionales e iterativas. [Usar] • Depurar programas con errores de sintaxis y lógicos. [Usar] • Diseñar e implementar algoritmos simples usando bucles. [Usar]
Lecturas : [Zel10], [Gut13], [BB19]	

Unidad 4: Funciones y paso de parámetros (10 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Definición e invocación de funciones. • Mecanismos de paso de parámetros (por valor, por referencia). • Valores de retorno y múltiples retornos. • Alcance y tiempo de vida de variables (local, global, nonlocal). • Fundamentos de recursión. • Expresiones lambda y funciones de orden superior. • Funciones como objetos de primera clase. • Documentación y prueba de funciones.	• Definir e implementar funciones con paso de parámetros adecuado. [Usar] • Explicar la diferencia entre paso por valor y paso por referencia. [Familiarizarse] • Implementar soluciones recursivas a problemas simples. [Usar] • Usar expresiones lambda para construcciones de programación funcional. [Usar] • Diseñar funciones con alcance y documentación apropiados. [Evaluar]
Lecturas : [Zel10], [Gut13], [HT99]	

Unidad 5: Estructuras de datos básicas (10 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Listas y operaciones con listas (indexación, segmentación, mutabilidad). • Tuplas e inmutabilidad. • Diccionarios y mapeo clave-valor. • Conjuntos y operaciones con conjuntos. • Cadenas como secuencias. • Comprensión de listas y expresiones generadoras. • Algoritmos básicos de búsqueda y ordenamiento. • Introducción a la complejidad algorítmica para operaciones básicas.	• Implementar programas usando listas, diccionarios y conjuntos. [Usar] • Elegir estructuras de datos apropiadas para problemas dados. [Evaluar] • Explicar las compensaciones entre diferentes estructuras de datos. [Familiarizarse] • Implementar operaciones básicas de búsqueda y ordenamiento en listas. [Usar]
Lecturas : [Zel10], [Gut13]	

Unidad 6: Programación Orientada a Objetos I: Fundamentos (10 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Subclases, herencia (incluyendo herencia múltiple) y sobrescritura de métodos. • Despacho dinámico: definición de llamada a método. • Idiomas orientados a objetos para encapsulación: – Privacidad, ocultación de datos y visibilidad de miembros de clase. – Interfaces que revelan solo firmas de métodos. – Clases base abstractas, rasgos (<i>traits</i>) y mixins. • Clases y objetos en Python. • Atributos y métodos. • Fundamentos de herencia y polimorfismo. • Encapsulamiento y ocultación de información.	• Construir una jerarquía de clases simple utilizando subclases que permita reutilizar código para subclases distintas [Usar] • Explicar la relación entre la herencia orientada a objetos (compartición de código y sobrescritura) y el subtipado (la idea de que un subtipo sea usable en un contexto que espera el supertipo) [Familiarizarse] • Comparar y contrastar los beneficios y costos/impacto de usar herencia (subclases) y composición (específicamente, cómo basar la composición en funciones de orden superior) [Evaluar] • Diseñar e implementar clases simples en Python. [Usar] • Explicar los beneficios de la POO para la organización del código. [Familiarizarse]
Lecturas : [Zel10], [Gut13]	

Unidad 7: Estrategias Algorítmicas (6 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Paradigmas: – Fuerza Bruta (por ejemplo, búsqueda lineal, ordenamiento por selección, viajante de comercio, mochila) – Disminuir y Vencer: * Por una Constante (por ejemplo, ordenamiento por inserción, ordenamiento topológico) * Por un Factor Constante (por ejemplo, búsqueda binaria) * Por un Tamaño Variable (por ejemplo, Euclides) – Dividir y Vencer (por ejemplo, búsqueda binaria, quicksort, mergesort, Strassen) – Voraz (por ejemplo, Dijkstra, Kruskal, Mochila) – Transformar y Vencer: * Simplificación de instancia (por ejemplo, encontrar duplicados mediante preordenamiento de lista) * Cambio de representación (por ejemplo, heapsort) * Reducción de problema (por ejemplo, mínimo común múltiplo, programación lineal) * Programación dinámica (por ejemplo, Floyd, Marshall, Bellman-Ford) – Compromisos espacio vs tiempo (por ejemplo, hash) • Iteración vs recursión (por ejemplo, factorial, búsqueda en árbol) • Estrategias de descomposición de problemas. • Técnicas de diseño de algoritmos. • Estrategias de depuración y prueba. • Seguimiento de la ejecución del programa.	• Para cada uno de los paradigmas en esta unidad: – Explicar sus características definitorias – Explicar un ejemplo que demuestre el paradigma incluyendo cómo este ejemplo satisface las características del paradigma. [Familiarizarse] • Dar ejemplos de algoritmos iterativos y recursivos que resuelvan el mismo problema, explicar los beneficios y desventajas de cada enfoque [Usar] • Evaluar si un enfoque voraz conduce a una solución óptima [Evaluar] • Aplicar técnicas sistemáticas de depuración para localizar y corregir errores. [Usar] • Analizar la complejidad algorítmica para problemas simples. [Familiarizarse]
Lecturas : [Gut13], [Cor+09]	

Unidad 8: Métodos Modernos de Desarrollo y Uso de Herramientas (4 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	Aprendizaje esperado (<i>Learning Outcomes</i>)
• Uso de entornos de programación modernos (IDEs, notebooks). • Mejores prácticas de búsqueda y reutilización de código. • Uso de bibliotecas estándar y APIs. • Técnicas de depuración y prueba. • Fundamentos de control de versiones. • Documentación de código y guías de estilo. • Entornos virtuales y gestión de paquetes.	• Usar herramientas de desarrollo modernas efectivamente. [Usar] • Aplicar principios de reutilización de código mediante bibliotecas y APIs. [Usar] • Depurar y probar programas sistemáticamente. [Usar] • Seguir estándares de codificación y prácticas de documentación. [Familiarizarse] • Gestionar paquetes y entornos de Python. [Familiarizarse]
Lecturas : [Zel10], [HT99]	

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. SISTEMA DE EVALUACIÓN

***** EVALUATION MISSING *****

10. BIBLIOGRAFÍA BÁSICA

- [HT99] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. 1st. Addison-Wesley, 1999.
- [Cor+09] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd. MIT Press, 2009.
- [Zel10] John M. Zelle. *Python Programming: An Introduction to Computer Science*. 2nd. Franklin, Beedle & Associates Inc, 2010.
- [Gut13] John V. Guttag. *Introduction to Computation and Programming Using Python*. 1st. MIT Press, 2013.
- [BB19] J. Glenn Brookshear and Dennis Brylow. *Computer Science: An Overview*. Global Edition. Pearson, 2019.