Introducción a la Programación: Fundamentos y Aplicaciones Básicas

Ciencias de la Educación | Licenciatura en tecnología e informática | para estudiantes universitarios | 4 semanas

Descripción del Curso

Este curso ofrece una introducción integral a los principios fundamentales de la programación, diseñado específicamente para estudiantes universitarios de la Licenciatura en Tecnología e Informática dentro del área de Ciencias de la Educación. A lo largo de cuatro semanas, los participantes explorarán conceptos esenciales como la lógica de programación, estructuras de datos básicas y la sintaxis de lenguajes de programación comunes.

El curso está dirigido a estudiantes que desean construir una base sólida en programación, sin requerir experiencia previa en codificación. Se adopta un enfoque metodológico práctico y didáctico que combina exposiciones teóricas con ejercicios aplicados y resolución de problemas para facilitar el aprendizaje activo y significativo.

Al finalizar el curso, los estudiantes serán capaces de comprender y aplicar las bases para escribir códigos funcionales y desarrollar algoritmos simples, sentando las bases para estudios avanzados en programación y desarrollo tecnológico en el ámbito educativo.

Objetivos Generales

- Comprender los fundamentos teóricos y prácticos de la programación.
- Aplicar técnicas básicas de codificación para desarrollar programas sencillos.
- Analizar problemas y diseñar algoritmos adecuados para su solución mediante programación.
- Evaluar y corregir errores comunes en códigos de programación.

Competencias

- Analizar y aplicar conceptos básicos de lógica de programación para resolver problemas simples.
- Interpretar y escribir código en un lenguaje de programación introductorio.
- Diseñar algoritmos básicos que reflejen soluciones a problemas cotidianos.
- Identificar y utilizar estructuras de control y datos fundamentales en programación.
- Desarrollar habilidades para depurar y corregir errores en códigos simples.
- Integrar conocimientos teóricos y prácticos para construir programas funcionales.

Requerimientos

- Conocimientos básicos de informática y manejo general de computadoras.

- Acceso a una computadora con un entorno de desarrollo integrado (IDE) recomendado (por ejemplo, Visual Studio Code, Python IDLE).
- Software de programación instalado según la plataforma utilizada (por ejemplo, Python, JavaScript).
- Conexión a internet para acceso a recursos complementarios y materiales en línea.

Unidades del Curso

Unidad 1: Fundamentos de la Programación y Lógica Computacional

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de explicar los conceptos fundamentales de la lógica computacional y su aplicación en la programación, identificando las estructuras básicas de control.
- Al finalizar la unidad, el estudiante será capaz de clasificar y describir los tipos de datos primarios utilizados en la programación, aplicándolos correctamente en la construcción de algoritmos simples.
- Al finalizar la unidad, el estudiante será capaz de diseñar algoritmos básicos utilizando estructuras de control condicionales y repetitivas, garantizando la resolución lógica de problemas planteados.
- Al finalizar la unidad, el estudiante será capaz de implementar pseudocódigos o diagramas de flujo que representen de manera clara y coherente algoritmos sencillos basados en la lógica computacional.
- Al finalizar la unidad, el estudiante será capaz de analizar y corregir errores lógicos en algoritmos básicos para mejorar la precisión y funcionalidad de los programas desarrollados.

Contenidos Temáticos

1. Introducción a la lógica computacional

- Concepto y relevancia de la lógica computacional en la programación: se explicará qué es la lógica computacional, su importancia para el desarrollo de algoritmos y su relación con la programación.
- Componentes básicos: proposiciones, conectores lógicos (AND, OR, NOT), tablas de verdad y equivalencias lógicas.
- Aplicación de la lógica en la toma de decisiones y control en programas.

2. Estructuras básicas de control

- Secuencias: ejecución lineal de instrucciones.
- Estructuras condicionales: if, if-else, switch/case (conceptos y uso).
- Estructuras repetitivas: bucles for, while, do-while (conceptos y uso).
- Importancia del control de flujo para la resolución de problemas.

3. Tipos de datos primarios en programación

- Definición y clasificación de tipos de datos.

- Tipos de datos básicos: enteros, reales (decimales), caracteres, booleanos.
- Representación y uso adecuado de cada tipo en algoritmos.
- Conversión entre tipos de datos y su impacto en la programación.

4. Diseño y representación de algoritmos básicos

- Definición y características de un algoritmo.
- Construcción de pseudocódigo: sintaxis básica, reglas y convenciones.
- Diagramas de flujo: símbolos estándar, elaboración y lectura.
- Relación entre pseudocódigo y diagramas de flujo para representar algoritmos.

5. Análisis y depuración de errores lógicos en algoritmos

- Tipos de errores en programación: sintácticos, semánticos y lógicos, con énfasis en errores lógicos.
- Metodologías para detectar y corregir errores lógicos en pseudocódigo y diagramas de flujo.
- Ejemplos prácticos de errores lógicos comunes y su corrección.
- Importancia de la revisión y prueba para mejorar la precisión y funcionalidad del algoritmo.

Actividades

Actividad 1: Análisis y construcción de tablas de verdad

Objetivo: Explicar los conceptos fundamentales de la lógica computacional y su aplicación en programación (Objetivo 1).

Descripción:

- Se proporcionará a los estudiantes una serie de proposiciones lógicas con conectores AND, OR y NOT.
- Los estudiantes crearán tablas de verdad completas para cada proposición.
- Se discutirán los resultados y su aplicación en estructuras condicionales.

Organización: Individual.

Producto esperado: Tablas de verdad elaboradas y análisis escrito breve.

Duración estimada: 1 hora.

Actividad 2: Clasificación y uso de tipos de datos en pseudocódigo

Objetivo: Clasificar y describir los tipos de datos primarios, aplicándolos correctamente en algoritmos simples (Objetivo 2).

Descripción:

- Se presentarán diferentes variables y valores a los estudiantes.
- En parejas, deberán identificar el tipo de dato correspondiente y escribir fragmentos de pseudocódigo que utilicen esas variables correctamente.
- Compartirán sus soluciones con el grupo para discusión.

Organización: Parejas.

Producto esperado: Fragmentos de pseudocódigo con uso correcto de tipos de datos.

Duración estimada: 1.5 horas.

Actividad 3: Diseño de algoritmos con estructuras de control

Objetivo: Diseñar algoritmos básicos utilizando estructuras condicionales y repetitivas para resolver problemas lógicos (Objetivo 3).

Descripción:

- Se plantearán problemas sencillos (por ejemplo, cálculo de promedio con condiciones, conteo de elementos que cumplen una condición).
- En grupos pequeños, los estudiantes diseñarán el pseudocódigo y el diagrama de flujo que resuelvan el problema usando estructuras condicionales y/o repetitivas.
- Presentarán sus diseños al grupo para retroalimentación.

Organización: Grupos de 3-4 estudiantes.

Producto esperado: Pseudocódigo y diagrama de flujo funcionales y claros.

Duración estimada: 2 horas.

Actividad 4: Identificación y corrección de errores lógicos en algoritmos

Objetivo: Analizar y corregir errores lógicos en algoritmos básicos para mejorar su precisión y funcionalidad (Objetivo 5).

Descripción:

- Se entregarán a los estudiantes varios pseudocódigos y diagramas de flujo con errores lógicos intencionales.
- Individualmente deberán identificar los errores, explicar su impacto y proponer correcciones.
- Se realizará una puesta en común para discutir las soluciones.

Organización: Individual.

Producto esperado: Informe con identificación, análisis y corrección de errores lógicos.

Duración estimada: 1.5 horas.

Evaluación

Evaluación diagnóstica

Qué se evalúa: Conocimientos previos sobre lógica y programación básica.

Cómo se evalúa: Cuestionario breve con preguntas conceptuales sobre lógica computacional, tipos de datos y estructuras de control.

Instrumento sugerido: Prueba escrita o formulario en línea con preguntas de opción múltiple y verdadero/falso.

Evaluación formativa

Qué se evalúa: Progreso en la aplicación de conceptos, diseño de algoritmos y corrección de errores.

Cómo se evalúa: Revisión continua de actividades prácticas (tablas de verdad, pseudocódigos, diagramas de flujo) con retroalimentación individual y grupal.

Instrumento sugerido: Rúbrica de evaluación para actividades prácticas, observación y discusión en clase.

Evaluación sumativa

Qué se evalúa: Dominio integral de los conceptos y habilidades de la unidad, incluyendo diseño, representación y corrección de algoritmos.

Cómo se evalúa: Proyecto final donde el estudiante deberá diseñar un algoritmo que resuelva un problema planteado, representarlo en pseudocódigo y diagrama de flujo, y justificar las decisiones tomadas, incluyendo la detección y corrección de posibles errores lógicos.

Instrumento sugerido: Rúbrica detallada para evaluar calidad del algoritmo, precisión del pseudocódigo y diagrama, y análisis de errores.

Unidad 2: Sintaxis y Estructuras de Control en Programación

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y describir la sintaxis básica de un lenguaje de programación introductorio, aplicando las reglas para la correcta escritura de código.
- Al finalizar la unidad, el estudiante será capaz de analizar y utilizar estructuras condicionales para controlar el flujo de ejecución en programas sencillos, implementando decisiones basadas en diferentes condiciones.
- Al finalizar la unidad, el estudiante será capaz de diseñar y ejecutar ciclos (bucles) para repetir instrucciones en la solución de problemas, garantizando la correcta repetición y terminación del proceso.
- Al finalizar la unidad, el estudiante será capaz de construir programas que integren sintaxis correcta con estructuras de control condicionales y cíclicas, evaluando y corrigiendo errores comunes para asegurar su funcionamiento adecuado.

Contenidos Temáticos

1. Introducción a la sintaxis básica en programación

- Definición y importancia de la sintaxis en programación: Concepto de sintaxis como conjunto de reglas que determinan la estructura correcta del código.
- Elementos básicos de un lenguaje de programación: Variables, tipos de datos, operadores, expresiones y sentencias.
- Reglas generales para la escritura del código: Identificadores, uso de mayúsculas y minúsculas, espacios, comentarios y terminadores de línea.
- Errores de sintaxis comunes: Tipos de errores y cómo identificarlos en el proceso de codificación.

2. Estructuras condicionales

- Concepto y función de las estructuras condicionales en el control de flujo.
- Tipos de estructuras condicionales básicas:
  - Sentencia `if`: Sintaxis y uso.
  - Sentencia `if-else`: Toma de decisiones binarias.
  - Sentencia `if-elif-else` o anidamiento: Decisiones múltiples.
  - Uso de operadores relacionales y lógicos en condiciones.
- Ejemplos prácticos de implementación de condicionales en programas sencillos.
- Errores comunes al usar condicionales y buenas prácticas.

3. Estructuras cíclicas (bucles)

- Concepto y propósito de los ciclos para la repetición de instrucciones.
- Tipos de ciclos:
  - Ciclo `while`: Condición al inicio.
  - Ciclo `for`: Iteración controlada por contador o colección.
- Sintaxis y ejemplos prácticos de cada tipo de ciclo.
- Control del ciclo: instrucciones `break` y `continue`.
- Problemas comunes: ciclos infinitos y cómo evitarlos.

4. Integración de sintaxis, condicionales y ciclos en programas

- Diseño de algoritmos que combinen estructuras condicionales y cíclicas.
- Construcción de programas completos con sintaxis correcta y estructuras de control.
- Depuración y corrección de errores comunes en programas integrados.
- Buenas prácticas para la escritura y lectura del código.

Actividades

Actividad 1: Identificación y corrección de errores de sintaxis

Objetivo: Identificar y describir la sintaxis básica aplicando reglas para la correcta escritura de código.

Descripción:

- Se entrega a los estudiantes fragmentos de código con errores de sintaxis intencionados.
- Los estudiantes analizan el código y señalan los errores encontrados.
- Proponen correcciones para que el código funcione correctamente.

Organización: Individual

Producto esperado: Informe breve con errores identificados y correcciones propuestas.

Duración estimada: 45 minutos

Actividad 2: Programando decisiones con estructuras condicionales

Objetivo: Analizar y utilizar estructuras condicionales para controlar el flujo de ejecución en programas sencillos.

Descripción:

- Se presenta un problema práctico (por ejemplo, determinar si un número es positivo, negativo o cero).
- Los estudiantes diseñan y escriben código utilizando condicionales para resolver el problema.
- Prueban el programa con diferentes datos de entrada y documentan los resultados.

Organización: Parejas

Producto esperado: Código funcional y reporte de pruebas.

Duración estimada: 1 hora

Actividad 3: Creando ciclos para repetir instrucciones

Objetivo: Diseñar y ejecutar ciclos para repetir instrucciones garantizando correcta repetición y terminación.

Descripción:

- Se plantea un problema que requiere repetir una tarea (por ejemplo, imprimir los números del 1 al 10).
- Los estudiantes implementan la solución usando ciclos `for` y `while`.
- Se discuten las diferencias y ventajas de cada ciclo en el contexto dado.

Organización: Grupos pequeños (3-4 estudiantes)

Producto esperado: Código de ambas soluciones y análisis comparativo.

Duración estimada: 1.5 horas

Actividad 4: Proyecto integrador de programación con estructuras de control

Objetivo: Construir programas que integren sintaxis correcta con estructuras condicionales y cíclicas, evaluando y corrigiendo errores.

Descripción:

- Se asigna un problema complejo que requiera tomar decisiones y repetir procesos (por ejemplo, un menú interactivo que permita al usuario seleccionar opciones repetidamente hasta decidir salir).
- Los estudiantes diseñan el algoritmo, escriben el código y lo prueban.
- Realizan una revisión entre pares para identificar posibles errores y sugerir mejoras.
- Entregan la versión final corregida y documentada.

Organización: Grupos de 4 estudiantes

Producto esperado: Programa funcional, documentación del proceso y reporte de revisión entre pares.

Duración estimada: 3 horas (puede desarrollarse en sesiones múltiples)

Evaluación

Evaluación diagnóstica

Qué se evalúa: Conocimientos previos sobre sintaxis básica y estructuras de control.

Cómo se evalúa: Cuestionario breve con preguntas de opción múltiple y preguntas abiertas sobre conceptos fundamentales.

Instrumento sugerido: Test escrito o en línea al inicio de la unidad.

Evaluación formativa

Qué se evalúa: Aplicación práctica de sintaxis, condicionales y ciclos durante las actividades.

Cómo se evalúa: Revisión continua de los productos parciales (códigos y reportes), retroalimentación directa y revisión entre pares.

Instrumento sugerido: Listas de cotejo para revisión de código, rúbrica para análisis de reportes y participación en discusiones.

Evaluación sumativa

Qué se evalúa: Capacidad para construir programas integrando sintaxis correcta y estructuras de control, y corregir errores.

Cómo se evalúa: Presentación y entrega del proyecto integrador, donde se evalúa el diseño, funcionalidad, corrección de errores y documentación.

Instrumento sugerido: Rúbrica detallada que considere aspectos técnicos, estructurales y de documentación.

Unidad 3: Algoritmos y Funciones Básicas

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de diseñar algoritmos simples que resuelvan problemas básicos utilizando diagramas de flujo o pseudocódigo.
- Al finalizar la unidad, el estudiante será capaz de implementar funciones y procedimientos en un lenguaje de programación, aplicando principios de modularidad para organizar el código.
- Al finalizar la unidad, el estudiante será capaz de analizar y optimizar algoritmos básicos para mejorar su eficiencia y claridad.
- Al finalizar la unidad, el estudiante será capaz de depurar y corregir errores comunes en funciones y procedimientos implementados.
- Al finalizar la unidad, el estudiante será capaz de evaluar la adecuación de diferentes estructuras de control y funciones para la solución de problemas específicos.

Contenidos Temáticos

1. Introducción a los Algoritmos

- Definición y características de un algoritmo: concepto, propiedades de finitud, claridad, entrada y salida, y efectividad.
- Importancia del diseño algorítmico en la programación y solución de problemas.
- Representación de algoritmos: diagramas de flujo y pseudocódigo.

2. Diseño de Algoritmos Simples

- Identificación del problema y análisis de requisitos.
- Construcción de diagramas de flujo básicos: símbolos estándar, secuencias, decisiones y bucles.
- Redacción de pseudocódigo para problemas simples: estructura, convenciones y claridad.
- Ejemplos prácticos de algoritmos para problemas cotidianos (cálculo de promedio, conversión de unidades, etc.).

3. Introducción a Funciones y Procedimientos

- Concepto de funciones y procedimientos: modularidad, reutilización y organización del código.
- Elementos de una función: nombre, parámetros, tipo de retorno y cuerpo.
- Ventajas de usar funciones: simplificación, mantenimiento y legibilidad del código.

4. Implementación de Funciones y Procedimientos en un Lenguaje de Programación

- Sintaxis básica para definir y llamar funciones y procedimientos.
- Parámetros: paso por valor y paso por referencia.
- Funciones con y sin valor de retorno.
- Ejemplos prácticos de implementación para problemas simples.

5. Análisis y Optimización de Algoritmos Básicos

- Criterios para evaluar la eficiencia: claridad, complejidad y legibilidad.
- Identificación de redundancias y mejoras en algoritmos simples.
- Refactorización de código para mejorar la modularidad y eficiencia.
- Buenas prácticas en el diseño y optimización de algoritmos.

6. Depuración y Corrección de Errores en Funciones y Procedimientos

- Errores comunes: sintácticos, lógicos y de ejecución en funciones y procedimientos.
- Técnicas de depuración: uso de mensajes de impresión, depuradores y pruebas unitarias simples.
- Metodologías para detectar y corregir errores en código modular.

7. Evaluación de Estructuras de Control y Funciones para Solución de Problemas

- Tipos de estructuras de control: secuenciales, condicionales y repetitivas.
- Criterios para seleccionar estructuras de control adecuadas según el problema.
- Comparación del uso de funciones versus procedimientos en distintos escenarios.
- Diseño de soluciones modulares combinando diferentes estructuras y funciones.

Actividades

Actividad 1: Diseño de Algoritmos con Diagramas de Flujo y Pseudocódigo

Objetivo: Diseñar algoritmos simples para resolver problemas básicos utilizando diagramas de flujo y pseudocódigo.

Descripción:

- Se presenta un problema cotidiano (por ejemplo, calcular el área de un rectángulo y su perímetro).
- Cada estudiante elabora un diagrama de flujo que represente el algoritmo para resolver el problema.
- Luego redactan el pseudocódigo correspondiente siguiendo las convenciones aprendidas.
- Se realiza una puesta en común para comparar y discutir diferentes diseños.

Organización: Individual

Producto esperado: Diagrama de flujo y pseudocódigo del algoritmo propuesto.

Duración estimada: 90 minutos

Actividad 2: Implementación de Funciones y Procedimientos

Objetivo: Implementar funciones y procedimientos en un lenguaje de programación aplicando modularidad.

Descripción:

- Se asigna un problema sencillo (por ejemplo, calcular el máximo de tres números).
- Los estudiantes escriben el código que defina funciones o procedimientos para resolver el problema.
- Se practica la llamada a funciones desde un programa principal y el manejo de parámetros.
- Se revisan y corrigen en parejas para detectar errores y mejorar el diseño.

Organización: Parejas

Producto esperado: Código funcional con funciones y procedimientos correctamente implementados.

Duración estimada: 2 horas

Actividad 3: Análisis y Optimización de Algoritmos

Objetivo: Analizar y optimizar algoritmos básicos para mejorar su eficiencia y claridad.

Descripción:

- Se entrega un algoritmo con código fuente que presenta redundancias o estructuras poco claras.
- Los estudiantes identifican oportunidades de mejora y proponen optimizaciones.
- Implementan las mejoras y comparan la versión original con la optimizada.
- Discuten en grupo los criterios utilizados para la optimización.

Organización: Grupos de 3-4 estudiantes

Producto esperado: Informe con análisis, código optimizado y justificación de cambios.

Duración estimada: 3 horas

Actividad 4: Depuración y Corrección de Errores en Funciones

Objetivo: Depurar y corregir errores comunes en funciones y procedimientos implementados.

Descripción:

- Se proporciona un conjunto de programas con funciones que contienen errores intencionales (sintaxis, lógica, parámetros).
- Los estudiantes ejecutan pruebas para identificar errores y utilizan técnicas de depuración.
- Corrigen los errores y validan el funcionamiento correcto del programa.
- Se realiza una reflexión escrita sobre los tipos de errores encontrados y estrategias usadas.

Organización: Individual

Producto esperado: Código corregido y documento con análisis de errores y soluciones.

Duración estimada: 2 horas

Evaluación

Evaluación Diagnóstica

Qué se evalúa: Conocimientos previos sobre algoritmos básicos, diagramas de flujo y funciones.

Cómo se evalúa: Cuestionario breve con preguntas teóricas y ejercicios simples para diseñar un algoritmo.

Instrumento sugerido: Prueba escrita o en línea con preguntas de opción múltiple y respuesta abierta.

Evaluación Formativa

Qué se evalúa: Progreso en el diseño, implementación, análisis y depuración de algoritmos y funciones.

Cómo se evalúa: Revisión continua de actividades prácticas, observación de participación, retroalimentación oral y escrita.

Instrumento sugerido: Rúbricas para actividades prácticas, registro de observaciones y retroalimentación personalizada.

Evaluación Sumativa

Qué se evalúa: Competencia en diseñar algoritmos con diagramas de flujo y pseudocódigo, implementar funciones y procedimientos, optimizar y depurar código, y seleccionar estructuras adecuadas.

Cómo se evalúa: Proyecto integrador que incluya el diseño, implementación y optimización de un programa con funciones y procedimientos para resolver un problema específico.

Instrumento sugerido: Entrega de proyecto con código fuente, documentación (diagramas y pseudocódigo), y presentación oral o escrita explicando las decisiones tomadas.

Unidad 4: Depuración, Pruebas y Aplicaciones Básicas

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y clasificar errores comunes en programas mediante la revisión sistemática del código.
- Al finalizar la unidad, el estudiante será capaz de aplicar técnicas básicas de depuración para corregir errores en programas utilizando herramientas de desarrollo integradas.
- Al finalizar la unidad, el estudiante será capaz de diseñar y ejecutar pruebas básicas de software para validar la funcionalidad de programas sencillos.
- Al finalizar la unidad, el estudiante será capaz de desarrollar aplicaciones básicas que integren estructuras de control, algoritmos y depuración, asegurando su correcto funcionamiento.
- Al finalizar la unidad, el estudiante será capaz de evaluar y justificar la corrección de un programa mediante la aplicación de pruebas y análisis de resultados.