

# Programación TIC: Fundamentos y Aplicaciones para la Educación Técnica

*Ciencias de la Educación | Licenciatura en tecnología e informática | para estudiantes de educación técnica/tecnológica | 32 semanas*

## Descripción del Curso

Este curso está diseñado para estudiantes de educación técnica y tecnológica que desean adquirir conocimientos sólidos en programación aplicada a las Tecnologías de la Información y Comunicación (TIC). Su propósito es proporcionar una base integral en conceptos, lenguajes y técnicas de programación, con un enfoque especial en el desarrollo de soluciones tecnológicas orientadas a la educación y la gestión informática.

El curso está dirigido a futuros profesionales en tecnología e informática que buscan desarrollar habilidades prácticas para diseñar, implementar y mantener aplicaciones informáticas que respondan a necesidades educativas y organizativas. Se emplea una metodología combinada que integra teoría, práctica y proyectos aplicados, favoreciendo el aprendizaje activo y colaborativo.

Al finalizar el curso, los estudiantes serán capaces de comprender los principios fundamentales de la programación, manejar diferentes lenguajes y herramientas TIC, y desarrollar aplicaciones básicas que contribuyan a mejorar procesos educativos y técnicos en entornos reales.

## Objetivos Generales

- Comprender y aplicar los principios fundamentales de la programación en el desarrollo de soluciones TIC.
- Diseñar, codificar y evaluar programas utilizando lenguajes de programación adecuados a contextos educativos y tecnológicos.
- Analizar y resolver problemas técnicos mediante la implementación de algoritmos eficientes y estructurados.
- Gestionar proyectos básicos de software siguiendo estándares de documentación y mantenimiento.
- Integrar tecnologías digitales para innovar en procesos educativos y administrativos.

## Competencias

- Analizar problemas y diseñar algoritmos eficientes para su solución mediante programación.
- Desarrollar aplicaciones utilizando lenguajes de programación adecuados al contexto TIC.
- Implementar y depurar código fuente para asegurar el correcto funcionamiento de programas informáticos.
- Aplicar buenas prácticas de documentación y mantenimiento de software en proyectos TIC.
- Integrar herramientas tecnológicas para la mejora de procesos educativos y administrativos.
- Trabajar colaborativamente en equipos multidisciplinarios para el desarrollo de proyectos tecnológicos.

## Requerimientos

- Conocimientos básicos en computación e informática general.
- Acceso a una computadora con ambiente de desarrollo integrado (IDE) instalado.
- Conexión a internet para acceso a recursos y plataformas educativas.
- Disposición para el trabajo colaborativo y aprendizaje autónomo.

## Unidades del Curso

### Unidad 1: Introducción a la Programación y TIC

#### Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y describir los conceptos básicos de programación, explicando su relevancia en el desarrollo de soluciones TIC en contextos educativos.
- Al finalizar la unidad, el estudiante será capaz de analizar la evolución histórica de las tecnologías de la información y comunicación (TIC), relacionando sus hitos clave con la transformación de la educación técnica.
- Al finalizar la unidad, el estudiante será capaz de explicar la importancia de las TIC en la educación técnica, ejemplificando su aplicación para mejorar procesos de enseñanza y aprendizaje.
- Al finalizar la unidad, el estudiante será capaz de diferenciar los tipos de lenguajes de programación básicos, evaluando su aplicabilidad en proyectos educativos y tecnológicos simples.
- Al finalizar la unidad, el estudiante será capaz de diseñar algoritmos sencillos que reflejen la lógica de programación fundamental, aplicando estructuras básicas para resolver problemas educativos.

#### Contenidos Temáticos

##### 1. Conceptos básicos de programación

- Definición de programación: explicación clara de qué es programar y su propósito.
- Elementos fundamentales de la programación: variables, tipos de datos, operadores, estructuras de control (secuenciales, condicionales, repetitivas).
- Relevancia de la programación en soluciones TIC: cómo la programación facilita el desarrollo de aplicaciones y herramientas educativas.

##### 2. Historia y evolución de las Tecnologías de la Información y Comunicación (TIC)

- Orígenes de las TIC: desde la comunicación tradicional hasta la digitalización.
- Hitos clave en la evolución de las TIC: aparición de computadoras, internet, dispositivos móviles, software educativo.

- Impacto de las TIC en la educación técnica: transformaciones en metodologías, acceso a recursos y formación profesional.

### **3. Importancia de las TIC en la educación técnica**

- Beneficios de integrar TIC en la enseñanza técnica: mejora en la comprensión, interacción y motivación.
- Aplicaciones prácticas de TIC en procesos educativos: simuladores, plataformas virtuales, recursos multimedia.
- Ejemplos de TIC en educación técnica: casos de éxito y experiencias relevantes.

### **4. Tipos de lenguajes de programación básicos**

- Lenguajes de bajo nivel: características y ejemplos básicos.
- Lenguajes de alto nivel: características, ejemplos comunes (Python, JavaScript, Scratch).
- Comparación y aplicabilidad: criterios para seleccionar un lenguaje según el proyecto educativo o tecnológico.

### **5. Diseño de algoritmos sencillos y lógica de programación fundamental**

- Concepto de algoritmo: definición y representación (pseudocódigo, diagramas de flujo).
- Estructuras básicas de programación: secuencia, decisión y repetición.
- Construcción de algoritmos para resolver problemas educativos: ejemplos prácticos y ejercicios guiados.

## **Actividades**

### **Actividad 1: "Mapa conceptual de conceptos básicos de programación"**

**Objetivo:** Identificar y describir los conceptos básicos de programación.

**Descripción:**

- El docente presenta brevemente los conceptos fundamentales de programación.
- Los estudiantes, en grupos de 3 personas, elaboran un mapa conceptual que incluya variables, tipos de datos, operadores y estructuras de control.
- Se realiza una puesta en común donde cada grupo explica su mapa y se discuten las conexiones entre los conceptos.

**Organización:** Grupos de 3 estudiantes.

**Producto esperado:** Mapa conceptual físico o digital que refleje los conceptos básicos de programación.

**Duración estimada:** 1 hora.

### **Actividad 2: "Línea de tiempo de la evolución de las TIC"**

**Objetivo:** Analizar la evolución histórica de las TIC y su impacto en la educación técnica.

**Descripción:**

- El docente explica los hitos principales en la evolución de las TIC.

- En parejas, los estudiantes investigan y elaboran una línea de tiempo que incluya los hitos más relevantes y su impacto en la educación técnica.
- Presentan su línea de tiempo al grupo y comentan cómo cada hito ha transformado la educación técnica.

**Organización:** Parejas.

**Producto esperado:** Línea de tiempo ilustrada, física o digital.

**Duración estimada:** 1.5 horas.

### **Actividad 3: "Análisis de aplicaciones TIC en educación técnica"**

**Objetivo:** Explicar la importancia de las TIC en la educación técnica con ejemplos prácticos.

**Descripción:**

- Se presentan distintas aplicaciones TIC utilizadas en educación técnica (plataformas virtuales, simuladores, recursos multimedia).
- En grupos de 4, los estudiantes seleccionan una aplicación y analizan cómo mejora un proceso de enseñanza-aprendizaje.
- Realizan una breve exposición y generan un informe con sus conclusiones.

**Organización:** Grupos de 4 estudiantes.

**Producto esperado:** Informe escrito y presentación oral.

**Duración estimada:** 2 horas.

### **Actividad 4: "Diseño y codificación de un algoritmo sencillo"**

**Objetivo:** Diseñar algoritmos sencillos aplicando lógica de programación fundamental.

**Descripción:**

- El docente introduce el concepto de algoritmo y las estructuras básicas (secuencia, decisión, repetición).
- Individualmente, los estudiantes diseñan un algoritmo en pseudocódigo o diagrama de flujo para resolver un problema educativo simple (por ejemplo, calcular la nota final de un estudiante).
- Opcionalmente, se codifica el algoritmo en un lenguaje de programación básico (como Scratch o Python simple).
- Se revisan y discuten los diseños y códigos en clase, identificando mejoras y buenas prácticas.

**Organización:** Individual.

**Producto esperado:** Algoritmo en pseudocódigo o diagrama de flujo y código fuente si aplica.

**Duración estimada:** 2 horas.

## **Evaluación**

### **Evaluación diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre programación, TIC y su aplicación en educación técnica.

**Cómo se evalúa:** Cuestionario breve con preguntas abiertas y de opción múltiple sobre conceptos básicos y experiencia previa con TIC.

**Instrumento sugerido:** Test escrito o digital con 10 preguntas.

### **Evaluación formativa**

**Qué se evalúa:** Progreso en la comprensión y aplicación de conceptos, participación en actividades y calidad de los productos entregados.

**Cómo se evalúa:** Observación directa, revisión de mapas conceptuales, líneas de tiempo, informes y algoritmos diseñados.

**Instrumento sugerido:** Rúbricas específicas para cada actividad que consideren claridad, precisión, aplicabilidad y creatividad.

### **Evaluación sumativa**

**Qué se evalúa:** Dominio integral de los objetivos de la unidad: descripción de conceptos, análisis histórico, explicación de importancia TIC, diferenciación de lenguajes y diseño de algoritmos.

**Cómo se evalúa:** Examen escrito y práctico con preguntas teóricas y ejercicios de diseño de algoritmos; presentación final de un proyecto simple que integre programación y TIC en un contexto educativo.

**Instrumento sugerido:** Examen escrito, evaluación práctica y rúbrica para presentación de proyecto.

## **Unidad 2: Algoritmos y Estructuras de Datos Básicas**

### **Objetivos de Aprendizaje**

- Al finalizar la unidad, el estudiante será capaz de analizar y describir algoritmos fundamentales para la resolución de problemas básicos utilizando diagramas de flujo y pseudocódigo.
- Al finalizar la unidad, el estudiante será capaz de implementar estructuras de datos simples, como arreglos y listas, en un lenguaje de programación adecuado para organizar y manipular información.
- Al finalizar la unidad, el estudiante será capaz de diseñar y codificar algoritmos que utilicen estructuras de datos básicas para resolver problemas técnicos específicos, evaluando su eficiencia y corrección.
- Al finalizar la unidad, el estudiante será capaz de depurar y optimizar programas que emplean algoritmos y estructuras de datos simples, aplicando buenas prácticas de documentación y mantenimiento.

### **Contenidos Temáticos**

#### **1. Introducción a los algoritmos**

- Definición y características de los algoritmos: concepto, finitud, claridad, entrada, salida y efectividad.
- Importancia de los algoritmos en la programación y la resolución de problemas técnicos.
- Representación de algoritmos: diagramas de flujo y pseudocódigo.

## 2. Diagramas de flujo

- Elementos básicos de un diagrama de flujo: símbolos estándar (inicio/fin, proceso, decisión, entrada/salida, conectores).
- Construcción de diagramas de flujo para problemas simples: secuencias, decisiones y ciclos.
- Práctica de interpretación de diagramas de flujo.

## 3. Pseudocódigo

- Características y ventajas del pseudocódigo.
- Estructura básica del pseudocódigo: variables, asignaciones, entrada/salida, condicionales, bucles.
- Escritura de pseudocódigo para problemas básicos.

## 4. Estructuras de datos básicas

- Concepto de estructura de datos y su importancia en la programación.
- Arreglos (vectores): definición, declaración, acceso y manipulación de elementos.
- Listas simples: definición, diferencias con arreglos, operaciones básicas (inserción, eliminación, recorrido).
- Comparación entre arreglos y listas para diferentes tipos de problemas.

## 5. Implementación de estructuras de datos en un lenguaje de programación

- Declaración e inicialización de arreglos y listas en un lenguaje adecuado (por ejemplo, Python o Java).
- Operaciones básicas: recorrer, modificar, insertar y eliminar elementos.
- Problemas prácticos de organización y manipulación de información utilizando estas estructuras.

## 6. Diseño y codificación de algoritmos con estructuras de datos básicas

- Integración de algoritmos y estructuras de datos para resolver problemas técnicos comunes.
- Evaluación de la eficiencia de algoritmos simples: análisis básico de la complejidad temporal.
- Ejemplos de problemas específicos: búsqueda lineal, ordenamiento básico (burbuja o selección).

## 7. Depuración y optimización de programas

- Buenas prácticas para la depuración de código: identificación y corrección de errores comunes.
- Optimización de algoritmos y estructuras de datos: simplificación y mejora del rendimiento.
- Documentación y mantenimiento del código: comentarios, estructura clara, estilos de programación.

## Actividades

### Actividad 1: Construcción de diagramas de flujo para problemas básicos

**Objetivo:** Analizar y describir algoritmos fundamentales utilizando diagramas de flujo.

**Descripción:**

- Presentar un problema sencillo (por ejemplo, cálculo del área de un rectángulo, determinación del mayor de dos números).
- Guiar a los estudiantes en la identificación de pasos secuenciales y condicionales necesarios.
- En parejas, diseñar un diagrama de flujo que represente el algoritmo para el problema propuesto.
- Compartir y discutir los diagramas de flujo en grupo para corregir y mejorar.

**Organización:** Parejas

**Producto esperado:** Diagrama de flujo completo y correcto para el problema planteado.

**Duración estimada:** 1.5 horas

## **Actividad 2: Escritura y ejecución de pseudocódigo para algoritmos básicos**

**Objetivo:** Describir algoritmos mediante pseudocódigo para resolver problemas básicos.

**Descripción:**

- Proponer problemas que involucren decisiones y ciclos simples (por ejemplo, cálculo de factorial, suma de números pares en un rango).
- Individualmente, los estudiantes escribirán el pseudocódigo que resuelva el problema.
- Realizar una revisión cruzada con un compañero para validar la lógica y claridad del pseudocódigo.
- Ejecutar el pseudocódigo en un entorno simulado o interpretarlo para verificar su corrección.

**Organización:** Individual y revisión en parejas

**Producto esperado:** Documento con pseudocódigo funcional y validado.

**Duración estimada:** 2 horas

## **Actividad 3: Implementación y manipulación de arreglos y listas en código**

**Objetivo:** Implementar estructuras de datos simples para organizar y manipular información.

**Descripción:**

- Introducir un lenguaje de programación (por ejemplo, Python).
- Ejercitar la creación de arreglos y listas, realizar operaciones básicas: inserción, recorrido, búsqueda.
- Resolver un problema práctico, como almacenar y mostrar calificaciones de estudiantes, calcular promedio, buscar un valor específico.
- Discutir en grupos las ventajas y limitaciones de las estructuras utilizadas.

**Organización:** Individual y discusión en grupos pequeños

**Producto esperado:** Programa funcional que manipule arreglos o listas, con código documentado.

**Duración estimada:** 3 horas

## **Actividad 4: Diseño, codificación y optimización de un algoritmo para un problema técnico**

**Objetivo:** Diseñar y codificar algoritmos con estructuras de datos básicas, evaluar eficiencia y corregir errores.

## Descripción:

- Presentar un problema técnico específico (por ejemplo, ordenar una lista de números usando método burbuja, búsqueda lineal en una lista).
- En grupos, diseñar el algoritmo en pseudocódigo o diagrama de flujo.
- Implementar el algoritmo en código.
- Probar el programa con diferentes datos, identificar errores o inefficiencias.
- Aplicar técnicas de depuración y optimización, documentar los cambios realizados.
- Presentar el programa final y explicar las mejoras implementadas.

**Organización:** Grupos de 3-4 estudiantes

**Producto esperado:** Código optimizado, documentación de mejoras y presentación grupal.

**Duración estimada:** 4 horas

## Evaluación

### Evaluación diagnóstica

**Qué se evalúa:** Conocimientos previos sobre algoritmos, diagramas de flujo, pseudocódigo y estructuras de datos básicas.

**Cómo se evalúa:** Mini cuestionario teórico-práctico con preguntas de opción múltiple y ejercicios cortos para construir diagramas de flujo y pseudocódigo simple.

**Instrumento sugerido:** Cuestionario escrito o en plataforma virtual al inicio de la unidad.

### Evaluación formativa

**Qué se evalúa:** Proceso de aprendizaje en actividades prácticas: construcción de diagramas, pseudocódigo, implementación de estructuras, depuración.

**Cómo se evalúa:** Revisión continua de productos de actividades (diagramas, pseudocódigos, códigos fuente), retroalimentación individual y grupal.

**Instrumento sugerido:** Rúbricas de evaluación para cada actividad práctica y registros de observación docente.

### Evaluación sumativa

**Qué se evalúa:** Competencia para diseñar, codificar, depurar y optimizar algoritmos con estructuras de datos básicas, y documentar adecuadamente el trabajo.

**Cómo se evalúa:** Proyecto final individual o grupal que incluya diseño en diagrama de flujo o pseudocódigo, implementación en código, pruebas, depuración, optimización y documentación.

**Instrumento sugerido:** Rúbrica detallada que valore la corrección lógica, calidad del código, eficiencia, documentación y presentación.

## Unidad 3: Fundamentos de Lenguajes de Programación

## Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y diferenciar los principales paradigmas de programación (imperativo, orientado a objetos, funcional y lógico) mediante ejemplos prácticos.
- Al finalizar la unidad, el estudiante será capaz de analizar las características y estructuras básicas de lenguajes de programación comunes en TIC, como Python, Java y C, aplicando criterios técnicos para seleccionar el lenguaje adecuado según el contexto educativo o tecnológico.
- Al finalizar la unidad, el estudiante será capaz de diseñar y codificar programas simples utilizando sintaxis y estructuras propias de un lenguaje de programación seleccionado, garantizando la correcta implementación de algoritmos básicos.
- Al finalizar la unidad, el estudiante será capaz de evaluar y comparar la eficiencia y aplicabilidad de diferentes lenguajes de programación en función de requerimientos específicos de proyectos educativos o técnicos.

## Contenidos Temáticos

### 1. Introducción a los Lenguajes de Programación

- Definición y propósito de los lenguajes de programación: Explicación del concepto y la importancia en el desarrollo de soluciones tecnológicas.
- Historia y evolución: Breve recorrido histórico desde los primeros lenguajes hasta los actuales.
- Clasificación general: Lenguajes de bajo nivel vs. alto nivel.

### 2. Paradigmas de Programación

- Concepto de paradigma de programación: Definición y relevancia para la elección de un lenguaje.
- Paradigma Imperativo:
  - Características principales: instrucciones secuenciales, variables, estructuras de control.
  - Ejemplos prácticos: código básico en C.
  - Aplicaciones comunes y ventajas.
- Paradigma Orientado a Objetos:
  - Principios fundamentales: clases, objetos, encapsulación, herencia, polimorfismo.
  - Ejemplos prácticos: estructura básica en Java.
  - Ventajas en el desarrollo de proyectos modulares y escalables.
- Paradigma Funcional:
  - Características: funciones puras, inmutabilidad, ausencia de efectos secundarios.
  - Ejemplos prácticos: uso de funciones en Python.
  - Beneficios en la programación concurrente y matemática.
- Paradigma Lógico:

- Fundamentos: hechos, reglas, consultas.
- Ejemplos prácticos: introducción a Prolog.
- Aplicaciones en inteligencia artificial y bases de conocimiento.

### **3. Características y Estructuras Básicas de Lenguajes Comunes en TIC**

- Python:
  - Sintaxis y tipado dinámico.
  - Estructuras básicas: variables, tipos de datos, condicionales, bucles, funciones.
  - Ventajas y ámbitos de aplicación.
- Java:
  - Sintaxis y tipado estático.
  - Estructuras básicas: clases, métodos, control de flujo.
  - Características orientadas a objetos y uso en aplicaciones móviles y empresariales.
- C:
  - Sintaxis y tipado estático.
  - Estructuras básicas: punteros, funciones, estructuras, control de memoria.
  - Uso en sistemas embebidos y programación de bajo nivel.
- Criterios para seleccionar un lenguaje según contexto:
  - Facilidad de aprendizaje y uso.
  - Requerimientos de desempeño y eficiencia.
  - Compatibilidad con plataformas y herramientas disponibles.
  - Objetivos educativos o técnicos del proyecto.

### **4. Diseño y Codificación de Programas Simples**

- Algoritmos básicos: definición y representación (pseudocódigo y diagramas de flujo).
- Implementación práctica en un lenguaje seleccionado (Python, Java o C):
  - Declaración de variables y tipos de datos.
  - Estructuras condicionales y repetitivas.
  - Funciones y modularidad.
  - Manejo básico de errores.
- Pruebas y depuración: técnicas para asegurar la correcta ejecución.

### **5. Evaluación y Comparación de Lenguajes de Programación**

- Criterios de evaluación:
  - Eficiencia: uso de recursos, velocidad de ejecución.

- Facilidad de mantenimiento y escalabilidad.
- Comunidad y soporte.
- Compatibilidad con proyectos educativos o técnicos específicos.
- Análisis comparativo entre Python, Java y C según casos de uso reales.
- Selección del lenguaje más adecuado para diferentes escenarios.

## Actividades

### Actividad 1: Identificación y Clasificación de Paradigmas

**Objetivo:** Desarrollar la capacidad para identificar y diferenciar los principales paradigmas de programación.

**Descripción:**

- Se proporcionarán fragmentos de código representativos de cada paradigma (imperativo, orientado a objetos, funcional y lógico).
- Los estudiantes analizarán cada fragmento, discutirán en grupo las características observadas y clasificarán el paradigma correspondiente.
- Finalmente, cada grupo presentará un resumen justificando su clasificación con ejemplos.

**Organización:** Grupos de 3-4 estudiantes.

**Producto esperado:** Presentación breve con clasificación y justificación de cada fragmento de código.

**Duración estimada:** 90 minutos.

### Actividad 2: Análisis Comparativo de Lenguajes

**Objetivo:** Analizar características y estructuras básicas de Python, Java y C, aplicando criterios técnicos para selección de lenguaje.

**Descripción:**

- Se entregará a los estudiantes una tabla con características técnicas de los tres lenguajes.
- En parejas, los estudiantes evaluarán escenarios educativos o técnicos propuestos y seleccionarán el lenguaje más adecuado, argumentando su elección.
- Se realizará un debate grupal para contrastar opiniones y conclusiones.

**Organización:** Parejas y discusión grupal.

**Producto esperado:** Informe escrito breve con selección y justificación del lenguaje para cada escenario.

**Duración estimada:** 120 minutos.

### Actividad 3: Programación de un Algoritmo Básico

**Objetivo:** Diseñar y codificar un programa simple utilizando sintaxis y estructuras propias de un lenguaje seleccionado.

**Descripción:**

- Cada estudiante elegirá uno de los lenguajes estudiados (Python, Java o C).
- Se les asignará un problema sencillo (por ejemplo, cálculo de promedio, conversión de unidades o manejo de listas básicas).
- Deberán diseñar el algoritmo en pseudocódigo y codificarlo en el lenguaje seleccionado, garantizando la correcta implementación.
- Se realizarán pruebas para verificar la funcionalidad y se corregirán errores detectados.

**Organización:** Individual.

**Producto esperado:** Código fuente funcional y documentación del algoritmo (pseudocódigo y explicación).

**Duración estimada:** 180 minutos.

#### **Actividad 4: Evaluación Comparativa de Eficiencia y Aplicabilidad**

**Objetivo:** Evaluar y comparar la eficiencia y aplicabilidad de diferentes lenguajes según requerimientos específicos.

**Descripción:**

- Se presentarán casos de estudio con necesidades técnicas o educativas concretas.
- En grupos, los estudiantes discutirán ventajas y desventajas de usar Python, Java o C en cada caso.
- Realizarán una presentación final proponiendo el lenguaje más adecuado y justificando su elección con base en criterios técnicos.

**Organización:** Grupos de 3-4 estudiantes.

**Producto esperado:** Presentación y reporte con análisis comparativo y recomendación.

**Duración estimada:** 120 minutos.

### **Evaluación**

#### **Evaluación Diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre paradigmas y lenguajes de programación.

**Cómo se evalúa:** Cuestionario corto con preguntas de opción múltiple y respuesta abierta sobre conceptos básicos y paradigmas.

**Instrumento sugerido:** Prueba escrita o formulario digital (quiz) al inicio de la unidad.

#### **Evaluación Formativa**

**Qué se evalúa:** Progreso en la identificación de paradigmas, análisis de lenguajes, diseño y codificación de programas, y capacidad de argumentación técnica.

**Cómo se evalúa:** Revisión continua de actividades prácticas, retroalimentación en presentaciones grupales e individuales, y seguimiento de avances en programación.

**Instrumento sugerido:** Rúbricas para actividades prácticas y guías de observación para intervenciones orales y debates.

## Evaluación Sumativa

**Qué se evalúa:** Competencia para identificar paradigmas, analizar y seleccionar lenguajes, diseñar y codificar programas simples, y evaluar comparativamente lenguajes según contextos.

**Cómo se evalúa:** Examen teórico-práctico que incluya:

- Preguntas sobre paradigmas y características de lenguajes.
- Ejercicio de programación con código a desarrollar o corregir.
- Ensayo corto o análisis crítico sobre selección de lenguaje para un caso dado.

**Instrumento sugerido:** Examen escrito y entrega de código fuente con documentación.

## Unidad 4: Programación Estructurada en Lenguajes Procedurales

### Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y describir las estructuras de control básicas (secuenciales, condicionales y repetitivas) en lenguajes procedurales para estructurar algoritmos eficientes.
- Al finalizar la unidad, el estudiante será capaz de diseñar y codificar funciones y procedimientos modulados que permitan la reutilización de código y mejoren la organización de programas en entornos educativos y tecnológicos.
- Al finalizar la unidad, el estudiante será capaz de implementar programas estructurados que integren estructuras de control y modularidad, aplicando buenas prácticas de documentación y mantenimiento.
- Al finalizar la unidad, el estudiante será capaz de analizar y corregir errores lógicos y sintácticos en programas procedurales mediante técnicas de depuración y pruebas sistemáticas.
- Al finalizar la unidad, el estudiante será capaz de evaluar la eficiencia y claridad de programas desarrollados con programación estructurada, proponiendo mejoras basadas en principios de diseño modular y legibilidad.

### Contenidos Temáticos

#### 1. Introducción a la Programación Estructurada

- Concepto y características de la programación estructurada.
- Ventajas frente a otros paradigmas de programación.
- Lenguajes procedurales comunes en educación técnica (Pascal, C, etc.).

#### 2. Estructuras de Control Básicas en Lenguajes Procedurales

- **Estructura Secuencial:** ejecución lineal de instrucciones.
- **Estructuras Condicionales:** if, if-else, switch/case.
  - Sintaxis y ejemplos prácticos.
  - Aplicaciones en toma de decisiones simples y múltiples.
- **Estructuras Repetitivas:** for, while, do-while.

- Sintaxis y diferencias entre estructuras.
- Uso adecuado según el contexto.
- Ejemplos y análisis de algoritmos con estas estructuras.

### **3. Funciones y Procedimientos: Diseño y Codificación Modular**

- Concepto de funciones y procedimientos en programación estructurada.
- Ventajas de la modularidad y reutilización de código.
- Sintaxis para definir y llamar funciones y procedimientos.
- Parámetros: por valor y por referencia.
- Ámbito de variables y retorno de valores.
- Ejemplos de funciones y procedimientos en entornos educativos y tecnológicos.

### **4. Integración de Estructuras de Control y Modularidad en Programas**

- Diseño de programas estructurados combinando estructuras de control y funciones.
- Buenas prácticas de documentación:
  - Comentarios claros y concisos.
  - Convenciones de nombres.
  - Organización del código para mantenimiento.
- Ejemplos de programas completos con modularidad y control estructurado.

### **5. Depuración y Pruebas Sistemáticas en Programas Procedurales**

- Tipos de errores: sintácticos, semánticos y lógicos.
- Técnicas de depuración:
  - Uso de mensajes de salida (prints/debuggers).
  - Pruebas unitarias y de integración.
  - Identificación y corrección de errores comunes.
- Documentación de pruebas y correcciones.

### **6. Evaluación de la Eficiencia y Claridad de Programas Estructurados**

- Principios de diseño modular y legibilidad del código.
- Criterios para evaluar eficiencia:
  - Reducción de redundancia.
  - Optimización de estructuras de control.
  - Uso adecuado de funciones y procedimientos.
- Propuestas de mejoras basadas en análisis de código.
- Ejercicios prácticos de revisión y mejora de programas.

## Actividades

### Actividad 1: Identificación y Uso de Estructuras de Control

**Objetivo:** Identificar y describir las estructuras de control básicas en lenguajes procedurales para estructurar algoritmos eficientes.

**Descripción:**

- El docente presenta varios fragmentos de código con diferentes estructuras de control.
- Los estudiantes analizan y clasifican cada fragmento (secuencial, condicional, repetitiva).
- En parejas, diseñan un algoritmo simple que incluya al menos una estructura condicional y una repetitiva.
- Codifican el algoritmo en un lenguaje procedural básico.

**Organización:** Parejas

**Producto esperado:** Algoritmo diseñado y código funcional que utilice estructuras de control básicas.

**Duración estimada:** 2 horas

### Actividad 2: Diseño y Codificación de Funciones y Procedimientos

**Objetivo:** Diseñar y codificar funciones y procedimientos modulados para reutilización y organización de código.

**Descripción:**

- El docente explica la definición y sintaxis de funciones y procedimientos.
- Los estudiantes diseñan funciones que resuelvan tareas específicas (por ejemplo, cálculo de promedio, conversión de unidades).
- Codifican las funciones y procedimientos en un programa que las utilice para resolver un problema mayor.
- Documentan cada función con comentarios claros.

**Organización:** Individual

**Producto esperado:** Programa estructurado con funciones/procedimientos documentados y funcionando correctamente.

**Duración estimada:** 3 horas

### Actividad 3: Implementación y Documentación de un Programa Estructurado

**Objetivo:** Implementar programas estructurados que integren estructuras de control y modularidad, aplicando buenas prácticas de documentación y mantenimiento.

**Descripción:**

- Los estudiantes diseñan un programa con un problema real o educativo propuesto (por ejemplo, gestión de notas o inventario).
- El programa debe incluir estructuras condicionales, repetitivas y funciones/procedimientos.
- Se enfatiza la escritura de comentarios y la organización del código para facilitar su mantenimiento.

**Organización:** Grupos de 3-4 estudiantes

**Producto esperado:** Programa funcional documentado con enfoque modular y control estructurado.

**Duración estimada:** 4 horas

#### **Actividad 4: Depuración, Pruebas y Mejora de Programas**

**Objetivo:** Analizar y corregir errores lógicos y sintácticos mediante técnicas de depuración y evaluar la eficiencia y claridad de programas.

**Descripción:**

- Se entrega a cada estudiante un programa con errores comunes (sintaxis, lógica, redundancia).
- Realizan pruebas sistemáticas para detectar y corregir errores.
- Analizan la claridad y eficiencia del código, y proponen mejoras basadas en principios de modularidad y legibilidad.
- Documentan los cambios realizados y justifican las mejoras.

**Organización:** Individual

**Producto esperado:** Programa corregido y mejorado con documentación de depuración y optimización.

**Duración estimada:** 3 horas

#### **Evaluación**

##### **Evaluación Diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre estructuras de control y conceptos básicos de programación procedimental.

**Cómo se evalúa:** Cuestionario corto con preguntas teóricas y ejercicios breves de identificación de estructuras.

**Instrumento sugerido:** Prueba escrita o test en línea con preguntas de opción múltiple y respuestas cortas.

##### **Evaluación Formativa**

**Qué se evalúa:** Progreso en la comprensión y aplicación de estructuras de control, diseño modular, documentación y depuración.

**Cómo se evalúa:** Revisión continua de actividades prácticas, retroalimentación en clase, y participación en discusiones y ejercicios.

**Instrumento sugerido:** Rubricas para evaluación de códigos entregados, listas de cotejo para documentación y pruebas, observación directa.

##### **Evaluación Sumativa**

**Qué se evalúa:** Capacidad para identificar estructuras, diseñar funciones, implementar programas estructurados, depurar y mejorar código.

**Cómo se evalúa:** Proyecto integrador que requiera desarrollar un programa estructurado completo con documentación, pruebas y propuesta de mejoras.

**Instrumento sugerido:** Rúbrica detallada que valore correctitud del código, uso adecuado de estructuras, modularidad, documentación, pruebas y mejoras propuestas.

## Unidad 5: Introducción a la Programación Orientada a Objetos

### Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de identificar y explicar los conceptos fundamentales de la programación orientada a objetos, como clases, objetos, atributos y métodos, mediante ejemplos prácticos.
- Al finalizar la unidad, el estudiante será capaz de diseñar diagramas de clases para modelar soluciones tecnológicas sencillas, aplicando principios de encapsulamiento, herencia y polimorfismo.
- Al finalizar la unidad, el estudiante será capaz de codificar programas básicos utilizando un lenguaje orientado a objetos, implementando estructuras de clases y objetos que resuelvan problemas específicos del contexto educativo o técnico.
- Al finalizar la unidad, el estudiante será capaz de evaluar y depurar programas orientados a objetos para garantizar su correcto funcionamiento y eficiencia, siguiendo buenas prácticas de documentación y mantenimiento.

### Contenidos Temáticos

#### 1. Fundamentos de la Programación Orientada a Objetos (POO)

- **Introducción a la POO:** Definición, historia y ventajas frente a otros paradigmas.
- **Conceptos básicos:** Clases, objetos, atributos y métodos.
- **Ejemplos prácticos:** Representación de objetos cotidianos y técnicos mediante clases y objetos.

#### 2. Principios fundamentales de la POO

- **Encapsulamiento:** Definición, importancia y mecanismos de visibilidad (público, privado, protegido).
- **Herencia:** Concepto, tipos de herencia, reutilización de código y ejemplos prácticos.
- **Polimorfismo:** Sobrecarga y sobreescritura de métodos, uso y ventajas en programación orientada a objetos.

#### 3. Diseño de diagramas de clases para modelar soluciones tecnológicas

- **Elementos de un diagrama de clases:** Clases, atributos, métodos, relaciones y multiplicidad.
- **Relaciones entre clases:** Asociación, agregación, composición y herencia en diagramas UML.
- **Creación de diagramas de clases:** Modelado de casos sencillos relacionados con contextos educativos y técnicos.
- **Aplicación práctica:** Uso de herramientas básicas para diseñar diagramas UML (software libre o en línea).

#### 4. Programación básica orientada a objetos

- **Lenguajes orientados a objetos comunes:** Introducción breve (Java, Python, C++).
- **Implementación de clases y objetos:** Sintaxis básica para definir clases, atributos, métodos y crear objetos.

- **Ejercicios prácticos:** Programas simples que utilizan clases para resolver problemas técnicos o educativos.

## 5. Evaluación y depuración de programas orientados a objetos

- **Identificación de errores comunes:** Errores de sintaxis, lógica y diseño en POO.
- **Técnicas de depuración:** Uso de depuradores, impresión de mensajes y pruebas unitarias básicas.
- **Buenas prácticas:** Documentación del código, comentarios claros y estructuración para mantenimiento.
- **Optimización y refactorización:** Mejoras simples para aumentar eficiencia y legibilidad del código.

### Actividades

#### Actividad 1: Identificación y explicación de conceptos POO con ejemplos

**Objetivo:** Identificar y explicar los conceptos fundamentales de la POO mediante ejemplos prácticos.

**Descripción:**

- El docente presenta varios objetos cotidianos (por ejemplo, un automóvil, una lámpara, un teléfono).
- Los estudiantes, en parejas, identifican atributos y métodos de cada objeto.
- Cada pareja comparte sus ejemplos y el docente guía la explicación relacionándolos con clases y objetos.

**Organización:** Parejas

**Producto esperado:** Listado de atributos y métodos de objetos cotidianos, con justificación.

**Duración estimada:** 45 minutos

#### Actividad 2: Diseño de un diagrama de clases para un problema técnico sencillo

**Objetivo:** Diseñar diagramas de clases aplicando encapsulamiento, herencia y polimorfismo.

**Descripción:**

- El docente presenta un problema sencillo (por ejemplo, modelar un sistema de gestión de dispositivos electrónicos).
- En grupos, los estudiantes identifican clases, atributos, métodos y relaciones.
- Usan una herramienta básica (como draw.io o papel) para crear el diagrama UML.
- Exponen su diagrama y reciben retroalimentación del docente y compañeros.

**Organización:** Grupos de 3-4 estudiantes

**Producto esperado:** Diagrama de clases UML completo y presentado.

**Duración estimada:** 90 minutos

#### Actividad 3: Codificación básica de un programa orientado a objetos

**Objetivo:** Codificar programas básicos utilizando clases y objetos para resolver problemas específicos.

**Descripción:**

- El docente proporciona un enunciado para programar (por ejemplo, creación de una clase “Estudiante” con atributos y métodos para mostrar información).

- Individualmente, los estudiantes escriben el código en el lenguaje elegido (Java o Python).
- Se prueban los programas y se comparten los resultados en clase para discutir mejoras.

**Organización:** Individual

**Producto esperado:** Código funcional que implemente clases y métodos.

**Duración estimada:** 2 horas

#### **Actividad 4: Evaluación y depuración de un programa orientado a objetos con errores intencionales**

**Objetivo:** Evaluar y depurar programas para garantizar funcionamiento y eficiencia, aplicando buenas prácticas.

**Descripción:**

- El docente entrega un código con errores comunes (sintaxis, lógica, diseño).
- En parejas, los estudiantes analizan el código, identifican errores y proponen correcciones.
- Realizan las correcciones y documentan los cambios realizados.

**Organización:** Parejas

**Producto esperado:** Código corregido, documentado y con explicación de mejoras.

**Duración estimada:** 90 minutos

#### **Evaluación**

##### **Evaluación diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre programación en general y nociones básicas de POO.

**Cómo se evalúa:** Cuestionario breve con preguntas de opción múltiple y definición de términos.

**Instrumento sugerido:** Prueba escrita o cuestionario digital al inicio de la unidad.

##### **Evaluación formativa**

**Qué se evalúa:** Progreso en la comprensión y aplicación de conceptos, diseño de diagramas, codificación y depuración.

**Cómo se evalúa:** Revisión continua de actividades en clase, retroalimentación durante ejercicios prácticos y observación de participación.

**Instrumento sugerido:** Rúbricas para actividades prácticas, listas de cotejo para diagramas y códigos, participación en discusiones.

##### **Evaluación sumativa**

**Qué se evalúa:** Dominio integral de los conceptos, diseño, implementación y depuración de programas orientados a objetos.

**Cómo se evalúa:** Proyecto final donde el estudiante debe diseñar un diagrama de clases, codificar un programa y presentar la documentación y depuración realizada.

**Instrumento sugerido:** Rúbrica de evaluación que incluya aspectos de diseño, código funcional, documentación y calidad del mantenimiento.

## **Unidad 6: Manejo de Archivos y Bases de Datos Básicas**

### **Objetivos de Aprendizaje**

- Al finalizar la unidad, el estudiante será capaz de identificar y describir los tipos de archivos y bases de datos básicas utilizados en aplicaciones TIC, aplicando conceptos fundamentales en contextos educativos y tecnológicos.
- Al finalizar la unidad, el estudiante será capaz de crear y manipular archivos de texto y binarios mediante un lenguaje de programación específico, asegurando la correcta gestión y almacenamiento de datos.
- Al finalizar la unidad, el estudiante será capaz de diseñar y ejecutar consultas básicas en bases de datos simples, utilizando comandos estándar para recuperar y actualizar información de manera eficiente.
- Al finalizar la unidad, el estudiante será capaz de implementar algoritmos para la gestión de datos en archivos y bases de datos, evaluando la eficiencia y aplicabilidad en soluciones TIC educativas.
- Al finalizar la unidad, el estudiante será capaz de documentar y mantener el código relacionado con el manejo de archivos y bases de datos, siguiendo estándares de desarrollo y buenas prácticas en proyectos tecnológicos.

### **Contenidos Temáticos**

#### **1. Introducción al Manejo de Archivos y Bases de Datos Básicas**

- Conceptos fundamentales de datos y su importancia en aplicaciones TIC.
- Diferenciación entre archivos y bases de datos: definición y usos.
- Aplicaciones educativas y tecnológicas del manejo de datos.

#### **2. Tipos de Archivos Utilizados en TIC**

- Archivos de texto: características, formatos comunes (.txt, .csv, .json).
- Archivos binarios: definición, ejemplos y usos en almacenamiento eficiente.
- Comparación entre archivos de texto y binarios: ventajas y desventajas.
- Casos de uso en contextos educativos y tecnológicos.

#### **3. Bases de Datos Básicas**

- Definición y tipos básicos de bases de datos: relacionales y no relacionales.
- Elementos básicos de bases de datos relacionales: tablas, registros, campos.
- Introducción a sistemas gestores de bases de datos (SGBD) simples como SQLite o MySQL.
- Conceptos de integridad, consistencia y normalización básica.

#### **4. Creación y Manipulación de Archivos mediante Programación**

- Introducción a la manipulación de archivos con un lenguaje de programación específico (por ejemplo, Python o Java).
- Funciones y métodos para crear, abrir, leer, escribir y cerrar archivos de texto.
- Gestión de archivos binarios: apertura, lectura y escritura.
- Manejo de excepciones y errores comunes en la manipulación de archivos.
- Buenas prácticas para el almacenamiento y gestión eficiente de datos en archivos.

## **5. Diseño y Ejecución de Consultas Básicas en Bases de Datos Simples**

- Introducción al lenguaje SQL: estructura y comandos básicos.
- Sentencias SELECT para recuperación de datos.
- Inserción, actualización y eliminación de datos con INSERT, UPDATE y DELETE.
- Filtros y condiciones en consultas básicas (WHERE, ORDER BY).
- Ejemplos prácticos con bases de datos simples aplicados a contextos educativos.

## **6. Algoritmos para la Gestión de Datos en Archivos y Bases de Datos**

- Concepto de algoritmo en la gestión de datos.
- Algoritmos básicos para búsqueda, ordenamiento y actualización de datos en archivos.
- Implementación de algoritmos para consultas y modificaciones en bases de datos.
- Evaluación de la eficiencia y complejidad de los algoritmos implementados.
- Aplicabilidad de algoritmos en soluciones TIC para la educación técnica.

## **7. Documentación y Mantenimiento del Código Relacionado con Archivos y Bases de Datos**

- Importancia de la documentación en proyectos tecnológicos.
- Estándares y formatos para documentar código (comentarios, manuales, README).
- Buenas prácticas en la organización y mantenimiento del código fuente.
- Control de versiones básico y uso de herramientas para mantenimiento.
- Ejemplos prácticos de documentación aplicada a proyectos de gestión de datos.

## **Actividades**

### **Actividad 1: Identificación y Clasificación de Tipos de Archivos y Bases de Datos**

**Objetivo:** Contribuir al objetivo de identificar y describir los tipos de archivos y bases de datos básicas.

#### **Descripción:**

- Se entregan ejemplos de diferentes archivos y bases de datos simples.
- Los estudiantes deben analizar y clasificar cada archivo o base de datos según su tipo (texto, binario, relacional, no relacional).
- Realizan una presentación breve explicando sus características y posibles usos en contextos TIC educativos.

**Organización:** Grupos de 3-4 estudiantes.

**Producto esperado:** Tabla comparativa y presentación explicativa.

**Duración estimada:** 1.5 horas.

## **Actividad 2: Creación y Manipulación de Archivos con Código**

**Objetivo:** Contribuir al objetivo de crear y manipular archivos de texto y binarios mediante programación.

### **Descripción:**

- Se proporciona un conjunto de instrucciones para programar en un lenguaje específico (ejemplo: Python) operaciones básicas sobre archivos de texto y binarios.
- Los estudiantes deben desarrollar un programa que cree archivos, escriba datos, lea información y actualice contenido.
- Se enfatiza en el manejo adecuado de errores y cierre de archivos.

**Organización:** Individual.

**Producto esperado:** Código funcional y reporte breve que explique el proceso y resultados.

**Duración estimada:** 3 horas.

## **Actividad 3: Diseño y Ejecución de Consultas SQL Básicas**

**Objetivo:** Contribuir al objetivo de diseñar y ejecutar consultas básicas en bases de datos simples.

### **Descripción:**

- Se entrega una base de datos simple predefinida (por ejemplo: base de datos de estudiantes con campos básicos).
- Los estudiantes deben realizar consultas SELECT, INSERT, UPDATE y DELETE para distintos escenarios propuestos.
- Se evaluará la correcta sintaxis y lógica de las consultas, así como la interpretación de los resultados.

**Organización:** Parejas.

**Producto esperado:** Archivo con sentencias SQL y resultados obtenidos documentados.

**Duración estimada:** 2.5 horas.

## **Actividad 4: Implementación y Evaluación de Algoritmos para Gestión de Datos**

**Objetivo:** Contribuir al objetivo de implementar algoritmos para la gestión de datos en archivos y bases de datos, evaluando eficiencia y aplicabilidad.

### **Descripción:**

- Se propone el desarrollo de algoritmos para búsqueda y actualización de datos en archivos o bases de datos simples.
- Los estudiantes deben implementar el algoritmo, medir su eficiencia básica (tiempo de ejecución o número de pasos) y reportar conclusiones.
- Se solicita discutir posibles mejoras y aplicaciones en contextos educativos.

**Organización:** Grupos de 3 estudiantes.

**Producto esperado:** Código del algoritmo, análisis de eficiencia y reporte escrito.

**Duración estimada:** 4 horas.

## **Actividad 5: Documentación y Mantenimiento de Código**

**Objetivo:** Contribuir al objetivo de documentar y mantener el código relacionado con manejo de archivos y bases de datos siguiendo buenas prácticas.

### **Descripción:**

- Se entrega código fuente desarrollado en actividades anteriores sin documentación.
- Los estudiantes deben añadir comentarios claros, estructurar el código para mejor legibilidad y preparar un manual básico de uso.
- Se enseña brevemente el uso de control de versiones para mantenimiento del código.

**Organización:** Individual.

**Producto esperado:** Código documentado y manual de usuario.

**Duración estimada:** 2 horas.

## **Evaluación**

### **Evaluación Diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre tipos de archivos, bases de datos y conceptos básicos de gestión de datos.

**Cómo se evalúa:** Cuestionario corto de opción múltiple y preguntas abiertas.

**Instrumento sugerido:** Test en línea o papel con 10 preguntas breves.

### **Evaluación Formativa**

**Qué se evalúa:** Progreso en la creación y manipulación de archivos, ejecución de consultas SQL, implementación de algoritmos y documentación del código.

**Cómo se evalúa:** Revisión continua de productos parciales, retroalimentación en actividades prácticas y autoevaluación guiada.

**Instrumento sugerido:** Listas de cotejo para actividades, rúbricas para código y reportes, observación directa del docente.

### **Evaluación Sumativa**

**Qué se evalúa:** Dominio integral de los objetivos de la unidad: identificación de tipos de datos, programación de archivos, consultas SQL, algoritmos y documentación.

**Cómo se evalúa:** Proyecto integrador donde el estudiante desarrolla una aplicación sencilla que gestione datos mediante archivos y base de datos, con documentación completa.

**Instrumento sugerido:** Rúbrica detallada que evalúe funcionalidad del código, corrección de consultas, eficiencia algorítmica y calidad de la documentación.

## **Unidad 7: Desarrollo de Interfaces y Aplicaciones Educativas**

### **Objetivos de Aprendizaje**

- Al finalizar la unidad, el estudiante será capaz de diseñar interfaces gráficas funcionales para aplicaciones educativas, aplicando principios de usabilidad y accesibilidad.
- Al finalizar la unidad, el estudiante será capaz de programar componentes interactivos en aplicaciones educativas utilizando un lenguaje de programación adecuado, garantizando la correcta interacción usuario-sistema.
- Al finalizar la unidad, el estudiante será capaz de evaluar y depurar interfaces y aplicaciones educativas mediante pruebas funcionales y de usabilidad para asegurar su calidad y eficacia.
- Al finalizar la unidad, el estudiante será capaz de integrar tecnologías digitales en el desarrollo de aplicaciones educativas para innovar en procesos de enseñanza-aprendizaje.
- Al finalizar la unidad, el estudiante será capaz de documentar el desarrollo de interfaces y aplicaciones educativas siguiendo estándares técnicos, facilitando su mantenimiento y actualización.

### **Contenidos Temáticos**

#### **1. Introducción al desarrollo de interfaces en aplicaciones educativas**

- Concepto y relevancia de las interfaces gráficas en el aprendizaje digital: se abordará la importancia de las interfaces para facilitar la interacción usuario-sistema en contextos educativos.
- Principios básicos de diseño centrado en el usuario: usabilidad, accesibilidad y experiencia de usuario (UX) aplicada a aplicaciones educativas.
- Herramientas y lenguajes de programación comunes para el desarrollo de interfaces educativas: introducción a entornos como HTML5, CSS, JavaScript, y frameworks básicos.

#### **2. Diseño de interfaces gráficas para aplicaciones educativas**

- Principios de usabilidad: consistencia, simplicidad, retroalimentación y control del usuario.
- Accesibilidad en interfaces educativas: normas WCAG, contraste visual, navegación con teclado, lectores de pantalla.
- Prototipado y diseño visual: uso de wireframes y mockups, herramientas para diseño gráfico (Figma, Adobe XD, etc.).
- Elementos básicos de la interfaz: botones, menús, formularios, iconos y tipografía.

#### **3. Programación de componentes interactivos en aplicaciones educativas**

- Lenguajes y entornos para programación de interfaces: introducción práctica a HTML, CSS y JavaScript para interacción básica.

- Eventos y manejo de eventos: clics, arrastrar y soltar, entrada de teclado, validación de formularios.
- Integración de multimedia y recursos interactivos: imágenes, audio, video y animaciones para mejorar la experiencia educativa.
- Uso de bibliotecas y frameworks ligeros para mejorar la interactividad (ejemplos: jQuery, Bootstrap).

#### **4. Evaluación y depuración de interfaces y aplicaciones educativas**

- Pruebas funcionales: verificación de que todos los componentes interactúan correctamente.
- Pruebas de usabilidad: métodos para evaluar la facilidad de uso y accesibilidad con usuarios reales o simulados.
- Identificación y corrección de errores comunes: uso de herramientas de depuración en navegadores y entornos de desarrollo.
- Documentación de errores y soluciones para mantenimiento.

#### **5. Integración de tecnologías digitales en el desarrollo de aplicaciones educativas**

- Incorporación de tecnologías emergentes: realidad aumentada, gamificación, inteligencia artificial básica.
- Uso de APIs y servicios externos para enriquecer aplicaciones educativas (ejemplos: Google Maps, servicios de traducción).
- Interoperabilidad y estándares en aplicaciones educativas: e-learning, SCORM, xAPI.

#### **6. Documentación técnica y mantenimiento de interfaces y aplicaciones educativas**

- Importancia de la documentación técnica para el mantenimiento y actualización.
- Estándares y formatos para documentación: diagramas, manuales de usuario, manuales técnicos.
- Buenas prácticas para escribir documentación clara y completa.
- Uso de herramientas para gestión de versiones y documentación (Git, Markdown).

### **Actividades**

#### **Diseño de prototipo de interfaz gráfica para una aplicación educativa**

**Objetivo:** Desarrollar la capacidad de diseñar interfaces gráficas funcionales aplicando principios de usabilidad y accesibilidad.

##### **Descripción:**

- Seleccionar un tema educativo para desarrollar una aplicación sencilla.
- Realizar un análisis de usuarios y definir requerimientos básicos.
- Crear wireframes y mockups utilizando herramientas como Figma o papel.
- Incluir aspectos de accesibilidad como contraste y navegación.
- Presentar el prototipo para retroalimentación entre compañeros.

**Organización:** Individual o en parejas.

**Producto esperado:** Prototipo visual detallado de la interfaz gráfica.

**Duración estimada:** 4 horas.

## **Programación de componentes interactivos básicos en una aplicación educativa**

**Objetivo:** Programar componentes interactivos garantizando la correcta interacción usuario-sistema.

### **Descripción:**

- Utilizar HTML, CSS y JavaScript para crear una pequeña aplicación educativa.
- Implementar botones, formularios y validaciones básicas.
- Agregar eventos para manejar interacciones (clics, entradas de datos).
- Incorporar multimedia simple (imágenes, audio) para enriquecer la experiencia.
- Probar la aplicación y corregir errores detectados.

**Organización:** Individual.

**Producto esperado:** Código funcional de una aplicación con componentes interactivos.

**Duración estimada:** 6 horas.

## **Evaluación de usabilidad y funcionalidad de una interfaz educativa**

**Objetivo:** Evaluar y depurar interfaces mediante pruebas funcionales y de usabilidad.

### **Descripción:**

- Seleccionar una aplicación educativa existente o desarrollada previamente.
- Diseñar un plan de pruebas funcionales y de usabilidad.
- Realizar pruebas con usuarios (compañeros) o mediante simulación.
- Registrar problemas detectados y sugerir mejoras.
- Presentar informe de evaluación con conclusiones y recomendaciones.

**Organización:** Grupos pequeños (3-4 personas).

**Producto esperado:** Informe de evaluación de usabilidad y funcionalidad.

**Duración estimada:** 4 horas.

## **Documentación técnica de un proyecto de aplicación educativa**

**Objetivo:** Documentar el desarrollo siguiendo estándares técnicos para facilitar mantenimiento y actualización.

### **Descripción:**

- Revisar el proyecto desarrollado en actividades previas.
- Elaborar documentación técnica que incluya: descripción general, diagramas, manual de usuario y manual técnico.
- Utilizar herramientas de documentación y control de versiones (por ejemplo, Markdown en repositorio Git).
- Compartir la documentación para revisión y retroalimentación.

**Organización:** Individual o parejas.

**Producto esperado:** Documentación técnica completa y organizada del proyecto.

**Duración estimada:** 3 horas.

## **Evaluación**

### **Evaluación diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre diseño y programación de interfaces, principios básicos de usabilidad y accesibilidad.

**Cómo se evalúa:** Cuestionario de opción múltiple y preguntas abiertas cortas al inicio de la unidad.

**Instrumento sugerido:** Test en línea o formulario impreso con preguntas clave.

### **Evaluación formativa**

**Qué se evalúa:** Progreso en diseño, programación, evaluación y documentación de interfaces durante el desarrollo de actividades.

**Cómo se evalúa:** Observación directa, revisión de prototipos, código fuente, informes de pruebas y documentación técnica.

**Instrumento sugerido:** Rúbricas específicas para cada actividad, retroalimentación escrita y oral.

### **Evaluación sumativa**

**Qué se evalúa:** Competencia integral para diseñar, programar, evaluar, integrar tecnologías y documentar interfaces educativas.

**Cómo se evalúa:** Proyecto final que incluya diseño, código funcional, evaluación de usabilidad y documentación técnica completa.

**Instrumento sugerido:** Rúbrica de evaluación global que contemple criterios de usabilidad, funcionalidad, integración tecnológica y calidad documental.

## **Unidad 8: Pruebas, Depuración y Mantenimiento de Software**

### **Objetivos de Aprendizaje**

- Al finalizar la unidad, el estudiante será capaz de identificar y clasificar errores comunes en programas informáticos mediante la aplicación de técnicas sistemáticas de prueba.
- Al finalizar la unidad, el estudiante será capaz de aplicar métodos de depuración para corregir fallas en el código utilizando herramientas y entornos de desarrollo adecuados.
- Al finalizar la unidad, el estudiante será capaz de diseñar y ejecutar casos de prueba que aseguren la funcionalidad y calidad del software conforme a especificaciones dadas.
- Al finalizar la unidad, el estudiante será capaz de implementar procedimientos de mantenimiento preventivo y correctivo en programas para garantizar su funcionamiento eficiente a lo largo del tiempo.
- Al finalizar la unidad, el estudiante será capaz de documentar los procesos de prueba, depuración y mantenimiento siguiendo estándares técnicos y buenas prácticas en proyectos de software.

## **Contenidos Temáticos**

### **1. Introducción a la Prueba, Depuración y Mantenimiento de Software**

- Concepto y importancia de la calidad del software.
- Relación entre pruebas, depuración y mantenimiento.
- Impacto de errores y fallas en el desarrollo y uso del software.

### **2. Identificación y Clasificación de Errores en Programas Informáticos**

- Tipos de errores: sintácticos, lógicos, de ejecución y de diseño.
- Errores comunes en programación: ejemplos y análisis.
- Técnicas sistemáticas para la identificación de errores.
- Herramientas básicas para detección de errores.

### **3. Métodos y Técnicas de Depuración de Software**

- Concepto y objetivo de la depuración.
- Depuración manual vs. depuración asistida por herramientas.
- Uso de entornos de desarrollo integrados (IDE) para depuración.
- Técnicas comunes: depuración por impresión (print debugging), puntos de interrupción, inspección de variables.
- Flujo sistemático para la localización y corrección de fallas.

### **4. Diseño y Ejecución de Casos de Prueba**

- Importancia de las pruebas para asegurar la calidad del software.
- Tipos de pruebas: unitarias, de integración, funcionales y de aceptación.
- Elementos de un caso de prueba: entrada, procedimiento, resultado esperado.
- Diseño de casos de prueba basados en especificaciones y requerimientos.
- Ejecución y registro de resultados de pruebas.
- Documentación y reporte de incidencias encontradas.

### **5. Mantenimiento de Software: Preventivo y Correctivo**

- Concepto y tipos de mantenimiento de software.
- Mantenimiento preventivo: actualización, optimización y documentación.
- Mantenimiento correctivo: identificación y corrección de fallas post-despliegue.
- Procedimientos para planificar y ejecutar mantenimiento eficiente.
- Buenas prácticas para garantizar la estabilidad y evolución del software.

### **6. Documentación de Procesos de Prueba, Depuración y Mantenimiento**

- Importancia de la documentación técnica en proyectos de software.

- Estándares y formatos recomendados para documentación.
- Elaboración de informes de prueba y depuración.
- Registro de cambios y mantenimiento en el software.
- Herramientas para la gestión y mantenimiento documental.

## Actividades

### Actividad 1: Análisis y Clasificación de Errores en Código Fuente

**Objetivo:** Identificar y clasificar errores comunes en programas informáticos mediante técnicas sistemáticas de prueba.

**Descripción:**

- Se proporcionará a los estudiantes un fragmento de código con errores sintácticos, lógicos y de ejecución.
- Los estudiantes deberán ejecutar el código y observar los comportamientos y mensajes de error.
- Aplicarán técnicas de revisión para identificar y clasificar cada error detectado.
- Presentarán un informe con la descripción de cada error, su clasificación y posible causa.

**Organización:** Individual

**Producto esperado:** Informe detallado de errores identificados y clasificados.

**Duración estimada:** 2 horas

### Actividad 2: Depuración Guiada Usando un Entorno de Desarrollo Integrado (IDE)

**Objetivo:** Aplicar métodos de depuración para corregir fallas en el código utilizando herramientas y entornos de desarrollo adecuados.

**Descripción:**

- Se proporcionará un proyecto de programación con errores para depurar.
- Los estudiantes usarán un IDE (por ejemplo, Visual Studio Code, Eclipse, o similar) para establecer puntos de interrupción, inspeccionar variables y ejecutar paso a paso.
- Realizarán las correcciones necesarias y verificaciones para garantizar la solución de los errores.
- Documentarán el proceso de depuración, incluyendo las herramientas y técnicas utilizadas.

**Organización:** Parejas

**Producto esperado:** Código corregido y documento con el proceso de depuración.

**Duración estimada:** 3 horas

### Actividad 3: Diseño y Ejecución de Casos de Prueba para un Módulo de Software

**Objetivo:** Diseñar y ejecutar casos de prueba que aseguren la funcionalidad y calidad del software conforme a especificaciones dadas.

**Descripción:**

- Se entregarán especificaciones funcionales para un módulo sencillo (por ejemplo, un sistema de registro o calculadora básica).
- Los estudiantes diseñarán casos de prueba detallados que cubran distintos escenarios, incluyendo casos normales y de borde.
- Ejecutarán las pruebas sobre el código proporcionado o desarrollado.
- Registrar los resultados y reportar cualquier fallo o comportamiento inesperado.

**Organización:** Grupos pequeños (3-4 estudiantes)

**Producto esperado:** Documento con casos de prueba, resultados y reporte de incidencias.

**Duración estimada:** 4 horas

#### **Actividad 4: Planificación y Ejecución de Mantenimiento Preventivo y Correctivo**

**Objetivo:** Implementar procedimientos de mantenimiento preventivo y correctivo en programas para garantizar su funcionamiento eficiente a lo largo del tiempo.

**Descripción:**

- Se entregará un proyecto de software con documentación básica y reportes de errores conocidos.
- Los estudiantes deberán planificar acciones de mantenimiento preventivo (mejoras, refactorización, actualización de documentación) y correctivo (corrección de errores reportados).
- Ejecutar las acciones planificadas y actualizar la documentación.
- Presentar un informe que incluya el plan, las acciones realizadas y resultados obtenidos.

**Organización:** Grupos pequeños

**Producto esperado:** Plan, código actualizado y documentación de mantenimiento.

**Duración estimada:** 4 horas

#### **Actividad 5: Elaboración de Documentación Técnica para Procesos de Prueba, Depuración y Mantenimiento**

**Objetivo:** Documentar los procesos de prueba, depuración y mantenimiento siguiendo estándares técnicos y buenas prácticas.

**Descripción:**

- Los estudiantes recogerán la información y resultados de las actividades previas de prueba, depuración y mantenimiento.
- Diseñarán documentos técnicos que incluyan: casos de prueba, informes de depuración, registro de cambios y procedimientos de mantenimiento.
- Se revisarán formatos estándar y se aplicarán buenas prácticas de redacción técnica.
- Compartirán y presentarán sus documentos para retroalimentación.

**Organización:** Individual o parejas

**Producto esperado:** Documentación técnica completa y estandarizada.

**Duración estimada:** 3 horas

## Evaluación

### Evaluación Diagnóstica

**Qué se evalúa:** Conocimientos previos sobre identificación de errores, técnicas básicas de depuración y conceptos de mantenimiento.

**Cómo se evalúa:** Cuestionario corto de selección múltiple y preguntas abiertas sobre conceptos claves.

**Instrumento sugerido:** Test en papel o plataforma digital (Google Forms, Moodle).

### Evaluación Formativa

**Qué se evalúa:** Desarrollo progresivo de habilidades en identificación de errores, depuración, diseño de pruebas, mantenimiento y documentación.

**Cómo se evalúa:** Revisión y retroalimentación continua de las actividades prácticas realizadas, participación en discusiones y entregas parciales.

**Instrumento sugerido:** Rubricas para actividades prácticas, listas de cotejo y observación directa.

### Evaluación Sumativa

**Qué se evalúa:** Competencia integral para identificar errores, aplicar depuración, diseñar y ejecutar casos de prueba, realizar mantenimiento y documentar procesos.

**Cómo se evalúa:** Proyecto final integrador en el que el estudiante corrige un programa con errores, diseña y ejecuta pruebas, documenta el proceso y propone un plan de mantenimiento.

**Instrumento sugerido:** Rubrica detallada que valore calidad técnica, exhaustividad del diagnóstico, corrección, documentación y presentación.

## Unidad 9: Integración de Tecnologías TIC en Proyectos Educativos

## Unidad 10: Proyecto Final de Programación TIC

### Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de diseñar un proyecto integrador de programación TIC que incluya análisis de requerimientos y planificación, siguiendo criterios de funcionalidad y usabilidad.
- Al finalizar la unidad, el estudiante será capaz de implementar una aplicación TIC utilizando lenguajes de programación adecuados, garantizando la correcta ejecución y eficiencia del código.
- Al finalizar la unidad, el estudiante será capaz de documentar el proyecto de software mediante informes técnicos y manuales de usuario, conforme a estándares de documentación establecidos.

- Al finalizar la unidad, el estudiante será capaz de evaluar y presentar su proyecto integrador, argumentando las soluciones implementadas y demostrando su aplicación en contextos educativos o tecnológicos.
- Al finalizar la unidad, el estudiante será capaz de aplicar técnicas básicas de gestión de proyectos para organizar las fases de desarrollo, controlando tiempos y recursos de manera eficiente.

## **Contenidos Temáticos**

### **1. Diseño del Proyecto Integrador de Programación TIC**

- **Análisis de requerimientos funcionales y no funcionales**
  - Identificación de necesidades y objetivos del proyecto
  - Definición de funcionalidades clave
  - Consideraciones de usabilidad y accesibilidad
- **Planificación del proyecto**
  - Definición de alcance y entregables
  - Elaboración de cronograma y asignación de tareas
  - Herramientas para la gestión y seguimiento del proyecto

### **2. Implementación de la Aplicación TIC**

- **Selección del lenguaje de programación y tecnologías**
  - Criterios para elegir lenguajes adecuados al proyecto
  - Uso de entornos de desarrollo integrados (IDE)
- **Desarrollo de código eficiente y funcional**
  - Buenas prácticas de programación
  - Control de versiones y manejo de errores
  - Pruebas unitarias y depuración

### **3. Documentación del Proyecto de Software**

- **Elaboración de informes técnicos**
  - Estructura y contenido de informes técnicos
  - Normas y formatos estandarizados
- **Manual de usuario**
  - Componentes y redacción de manuales de usuario
  - Ilustraciones y ejemplos prácticos

### **4. Evaluación y Presentación del Proyecto**

- **Preparación de la presentación**

- Organización del contenido para la defensa del proyecto
- Uso de soportes visuales y medios tecnológicos

- **Argumentación y demostración práctica**

- Explicación de soluciones implementadas
- Demostración en tiempo real de la aplicación

## **5. Gestión Básica de Proyectos para Programación TIC**

- **Fases del ciclo de vida del proyecto**

- Inicio, planificación, ejecución, control y cierre

- **Control de tiempos y recursos**

- Herramientas para seguimiento y ajuste de cronogramas
- Gestión eficiente de recursos humanos y materiales

## **Actividades**

### **Actividad 1: Análisis y Planificación del Proyecto TIC**

**Objetivo:** Diseñar un proyecto integrador con análisis de requerimientos y planificación, siguiendo criterios de funcionalidad y usabilidad.

**Descripción:**

- El estudiante selecciona un problema o necesidad relacionada con el contexto educativo o tecnológico.
- Realiza un análisis detallado de requerimientos funcionales y no funcionales, identificando funcionalidades clave y aspectos de usabilidad.
- Elabora un plan de proyecto que incluya alcance, cronograma y asignación de tareas, utilizando herramientas digitales para la gestión.

**Organización:** Individual o en parejas

**Producto esperado:** Documento de análisis de requerimientos y plan de proyecto con cronograma.

**Duración estimada:** 4 horas

### **Actividad 2: Desarrollo e Implementación de la Aplicación TIC**

**Objetivo:** Implementar una aplicación TIC usando lenguajes de programación adecuados, garantizando ejecución correcta y eficiencia del código.

**Descripción:**

- El estudiante programa la aplicación conforme al diseño anterior, aplicando buenas prácticas y documentando el código.

- Realiza pruebas unitarias y depuración para asegurar la funcionalidad y eficiencia.
- Utiliza herramientas de control de versiones para gestionar el desarrollo.

**Organización:** Individual

**Producto esperado:** Código fuente funcional y documentado de la aplicación TIC.

**Duración estimada:** 10 horas

### **Actividad 3: Elaboración de la Documentación Técnica y Manual de Usuario**

**Objetivo:** Documentar el proyecto mediante informes técnicos y manuales de usuario conforme a estándares.

**Descripción:**

- Redactar el informe técnico que describa el proyecto, su desarrollo y resultados.
- Crear un manual de usuario claro y didáctico que facilite el uso de la aplicación.
- Revisar y ajustar la documentación para asegurar claridad y coherencia.

**Organización:** Individual

**Producto esperado:** Informe técnico y manual de usuario en formato digital.

**Duración estimada:** 5 horas

### **Actividad 4: Presentación y Defensa del Proyecto Integrador**

**Objetivo:** Evaluar y presentar el proyecto argumentando soluciones y demostrando aplicación práctica.

**Descripción:**

- Preparar una presentación multimedia que resuma el diseño, desarrollo y resultados del proyecto.
- Realizar una exposición oral frente a compañeros y docente, demostrando la aplicación y respondiendo preguntas.
- Recibir retroalimentación para mejorar futuras implementaciones.

**Organización:** Individual o en parejas

**Producto esperado:** Presentación multimedia y defensa oral del proyecto.

**Duración estimada:** 3 horas

## **Evaluación**

### **Evaluación Diagnóstica**

**Qué se evalúa:** Conocimientos previos sobre análisis de requerimientos, planificación y lenguajes de programación.

**Cómo se evalúa:** Cuestionario escrito o en línea con preguntas teóricas y prácticas básicas.

**Instrumento sugerido:** Test diagnóstico con preguntas de opción múltiple y respuesta corta.

### **Evaluación Formativa**

**Qué se evalúa:** Progreso en el análisis de requerimientos, implementación del código, elaboración de documentación y uso de herramientas de gestión.

**Cómo se evalúa:** Revisión continua de productos parciales, retroalimentación en talleres y sesiones de seguimiento.

**Instrumento sugerido:** Rúbricas para análisis de requerimientos, calidad del código, documentación y participación en actividades.

### **Evaluación Sumativa**

**Qué se evalúa:** Calidad integral del proyecto integrador, incluyendo diseño, implementación, documentación y presentación.

**Cómo se evalúa:** Evaluación final del proyecto con presentación oral y entrega de todos los productos.

**Instrumento sugerido:** Rúbrica de evaluación final con criterios para aspectos técnicos, funcionales, documentales y comunicativos.