Python en acción: crea tu propia aplicación para resolver un problema real

Tecnología e Informática | Informática | Aprendizaje Basado en Proyectos

Descripción

En este proyecto de Aprendizaje Basado en Proyectos, los estudiantes aprenderán los fundamentos de Python mientras diseñan y construyen una aplicación funcional relacionada con una situación cercana a su vida escolar. El producto final será una aplicación de consola llamada **“Mi Aula Inteligente”**, capaz de registrar datos, procesarlos y ofrecer recomendaciones o resultados útiles. Cada equipo podrá elegir un enfoque, como organizar tareas académicas, controlar gastos personales, registrar hábitos saludables o calcular acciones para cuidar el ambiente.

A lo largo de seis sesiones, los estudiantes pasarán de analizar un problema cotidiano a planificar, programar, probar, corregir y presentar una solución. Aprenderán a utilizar variables, tipos de datos, entrada y salida de información, operadores, estructuras condicionales, ciclos, listas, diccionarios, funciones y archivos de texto. También practicarán la lectura de errores, la depuración, la colaboración y la comunicación de decisiones técnicas.

La propuesta conecta la programación con situaciones reales: una aplicación no se construye solamente escribiendo código, sino comprendiendo una necesidad, organizando información y probando si la solución funciona para otras personas. El docente actuará como guía y facilitador, mientras los estudiantes tomarán decisiones sobre el diseño de su proyecto, distribuirán responsabilidades y mantendrán un portafolio con sus avances, dificultades y aprendizajes.

Objetivos de Aprendizaje

- **Analizar** un problema cotidiano y convertirlo en una propuesta de solución programable mediante Python.
- **Aplicar** variables, tipos de datos, operadores, entradas, salidas, condicionales, ciclos, listas y diccionarios en programas funcionales.
- **Diseñar** algoritmos y funciones que organicen el funcionamiento de una aplicación de consola.
- **Depurar** programas identificando errores de sintaxis, lógica y ejecución, y documentando las correcciones realizadas.
- **Crear y presentar** colaborativamente una aplicación en Python que responda a una necesidad real y explicar las decisiones tomadas.

Recursos Necesarios

- Un computador por pareja o por estudiante, con Python 3 instalado.
- Editor de código: Thonny, Visual Studio Code o IDLE.
- Acceso a internet para consultar la documentación oficial de Python, si está disponible.

- Proyector o pantalla para demostraciones del docente.
- Tablero, marcadores y notas adhesivas.
- Seis hojas de papel tamaño carta por equipo para diseñar algoritmos, pruebas y reflexiones.
- Guía impresa de sintaxis básica de Python y tabla de errores frecuentes.
- Plantilla impresa para definir problema, usuarios, entradas, procesos y salidas.
- Plantilla de diagrama de flujo o pseudocódigo.
- Carpeta digital por equipo en Google Drive, OneDrive o dispositivo local.
- Video corto, de tres a cinco minutos, sobre aplicaciones creadas con Python y resolución de problemas.
- Rúbrica de proyecto, lista de cotejo de código, formato de coevaluación y diario de aprendizaje.
- Datos ficticios para probar las aplicaciones: tareas, gastos, hábitos, residuos o consumo de agua.

Requisitos Previos

- Reconocer operaciones básicas con números y utilizar expresiones sencillas.
- Manejar el teclado, crear carpetas, guardar archivos y abrir un programa.
- Comprender instrucciones secuenciales y representar procesos mediante pasos ordenados.
- Haber trabajado previamente nociones de algoritmo, problema, dato y resultado en Informática.
- Participar en equipos, escuchar ideas, asumir responsabilidades y respetar turnos de trabajo.
- No es necesario haber programado previamente; las actividades parten de ejemplos sencillos y progresivos.

Actividades

Sesión 1: De un problema cotidiano a las primeras líneas de Python

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Comprender que programar consiste en dar instrucciones claras para resolver una necesidad y comenzar a reconocer la estructura básica de un programa en Python.

Activación de conocimientos previos y enganche

- **Docente, 5 minutos:** Presenta la pregunta: “Si tuvieras que explicar a un robot cómo preparar tu mochila para mañana, ¿qué instrucciones exactas le darías?”. Escribe tres respuestas en el tablero y pregunta: “¿Qué ocurriría si una instrucción fuera ambigua o estuviera desordenada?”.
- **Docente, 3 minutos:** Muestra un video corto sobre una aplicación sencilla creada con Python. Después pregunta: “¿Qué problema resuelve esta aplicación y qué datos necesita?”.
- **Estudiantes, 2 minutos:** Responden en una nota adhesiva qué situación escolar les gustaría mejorar mediante una aplicación.

Contextualización: El docente explica que Python se usa en automatización, análisis de datos, videojuegos, inteligencia artificial y aplicaciones web. Los estudiantes relacionan la programación con organizar sus tareas, controlar gastos, registrar hábitos o tomar decisiones ambientales.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Presentación del contenido

Docente, 15 minutos: Presenta Python mediante el ejemplo `print("Hola, programador")`. Explica que un programa es una secuencia de instrucciones, que Python distingue mayúsculas y minúsculas y que los mensajes de error son pistas para corregir. Demuestra cómo abrir el editor, crear la carpeta del proyecto, guardar un archivo como `app_aula.py` y ejecutar el código.

Actividad 1. Exploradores de problemas

Objetivo: Analizar un problema cotidiano y convertirlo en una propuesta programable.

Tiempo: 60 minutos. **Organización:** Equipos de tres o cuatro.

- El docente entrega la plantilla con las preguntas: “¿Qué problema observamos?”, “¿A quién afecta?”, “¿Qué información debe ingresar el usuario?”, “¿Qué cálculo o decisión realizará el programa?” y “¿Qué resultado mostrará?”.
- Cada equipo elige un tema: tareas, gastos, hábitos saludables, reciclaje o consumo de agua.
- Los estudiantes entrevistan durante diez minutos a dos compañeros de otros equipos para comprobar si el problema es real y comprensible.
- El equipo redacta una propuesta de una página y la presenta en un minuto.

Docente: Pregunta: “¿El problema puede resolverse con datos concretos?”, “¿Qué sería una entrada y qué sería una salida?”. Ayuda a reducir propuestas demasiado amplias.

Evidencia: Ficha del problema con usuarios, entradas, procesos y salidas.

Actividad 2. Laboratorio de instrucciones básicas

Objetivo: Aplicar salida de información, variables, tipos de datos y entrada del usuario.

Tiempo: 70 minutos. **Organización:** Parejas.

- El docente escribe: `nombre = input("¿Cómo te llamas? ")` y `print("Hola,", nombre)`.
- Los estudiantes reproducen el ejemplo y cambian el mensaje por uno relacionado con su proyecto.
- Prueban variables de texto y número: `edad = int(input("Edad: "))` y `print(edad + 1)`.
- Resuelven el reto: solicitar nombre, actividad preferida y cantidad relacionada con el proyecto, y mostrar una frase personalizada.
- Cada pareja intercambia su programa con otra y verifica si las preguntas son claras.

Docente: Observa el uso de comillas, paréntesis e indentación. Pregunta: “¿Qué tipo de dato devuelve `input()`?”, “¿Por qué usamos `int()`?”.

Evidencia: Programa inicial ejecutable y captura de pantalla.

Actividad 3. Primer boceto del proyecto

Objetivo: Diseñar la interacción básica de la aplicación.

Tiempo: 60 minutos. **Organización:** Equipos.

- El equipo escribe el diálogo entre usuario y aplicación: bienvenida, preguntas, procesamiento y resultados.
- Representa el flujo en tarjetas con las palabras “Inicio”, “Entrada”, “Proceso”, “Salida” y “Fin”.
- Transforma las tarjetas en pseudocódigo con instrucciones numeradas.
- El docente revisa cada boceto y autoriza el paso a la siguiente sesión.

Evidencia: Guion de interacción y pseudocódigo inicial.

Diferenciación: Quienes necesitan apoyo reciben código incompleto con espacios para completar y una tarjeta visual de tipos de datos. Quienes terminan antes agregan validaciones sencillas, como solicitar nuevamente un dato vacío.

Transición: El docente indica: “Ya sabemos qué problema resolver y qué información necesitamos. En la próxima sesión enseñaremos al programa a tomar decisiones”.

Fase de Cierre

Tiempo estimado: 10 minutos.

Síntesis y reflexión

- **Docente, 5 minutos:** Solicita un “ticket de salida” con tres frases: “Hoy aprendí que…”, “Un error que pude corregir fue…” y “Mi programa necesita recibir como entrada…”.
- **Estudiantes, 3 minutos:** Responden: “¿Qué diferencia hay entre una instrucción y un problema?”, “¿Qué dato debe ingresar el usuario de mi aplicación?” y “¿Qué parte del código entiendo mejor?”.
- **Docente, 2 minutos:** Retroalimenta dos aciertos comunes y anticipa el uso de decisiones con `if`, `elif` y `else`.

Tarea: Preguntar a una persona en casa qué función le gustaría que tuviera una aplicación relacionada con el problema elegido.

Sesión 2: Programas que toman decisiones

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Utilizar comparaciones y estructuras condicionales para que la aplicación responda de manera diferente según los datos ingresados.

- **Docente, 5 minutos:** Recupera dos pseudocódigos de la sesión anterior y pregunta: “¿En qué momento nuestra aplicación debe mostrar una recomendación diferente?”.
- **Estudiantes, 3 minutos:** Resuelven oralmente: “Si la nota es mayor o igual a 3.0, mostrar aprobado; si no, mostrar necesita mejorar”.
- **Docente, 2 minutos:** Presenta el reto: lograr que el programa no entregue siempre la misma respuesta.

Contextualización: Se explica que muchas aplicaciones toman decisiones: un sistema recomienda una actividad, alerta sobre un gasto o indica si se cumplió una meta.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Presentación del contenido

Docente, 15 minutos: Modela el uso de operadores de comparación (>, , >=, ==) y la estructura:

```
if cantidad >= meta:
print("Meta cumplida")
else:
print("Aún puedes mejorar")
```

Explica la importancia de los dos puntos y la sangría, y demuestra una condición con tres resultados usando `elif`.

Actividad 1. Tarjetas de decisiones

Objetivo: Comparar datos y representar decisiones mediante algoritmos.

Tiempo: 55 minutos. **Organización:** Equipos.

- El docente entrega tarjetas con situaciones: “gasto mayor que presupuesto”, “tareas pendientes igual a cero”, “residuos reciclados mayor que residuos comunes”.
- Los equipos ordenan cada situación en tres columnas: dato, comparación y respuesta.
- Escriben el pseudocódigo de una situación usando “Si... entonces... si no...”.
- Un estudiante explica la decisión y otro la representa en un diagrama de flujo.

Docente: Pregunta: “¿Qué sucede si el dato es exactamente igual a la meta?”, “¿La condición cubre todos los casos?”.

Evidencia: Tres algoritmos condicionales y un diagrama.

Actividad 2. Código con decisiones

Objetivo: Aplicar `if`, `elif` y `else` en un programa funcional.

Tiempo: 75 minutos. **Organización:** Parejas.

- Los estudiantes crean un archivo de práctica y escriben un programa que solicite una cantidad.
- Implementan tres rangos, por ejemplo: bajo, medio y alto.
- Prueban valores ubicados por debajo, dentro y por encima de cada rango.

- Registran en una tabla el valor ingresado, la respuesta esperada y la respuesta obtenida.
- Corrigen al menos un error intencional colocado por el docente.

Docente: Pregunta: “¿Qué línea se ejecuta cuando la primera condición es falsa?”, “¿Qué diferencia hay entre = y == ?”.

Evidencia: Programa condicional y tabla de pruebas.

Actividad 3. Integración al proyecto

Objetivo: Incorporar decisiones al prototipo de la aplicación.

Tiempo: 60 minutos. **Organización:** Equipos.

- El equipo revisa su pseudocódigo y marca con color las partes que requieren una decisión.
- Convierte al menos dos decisiones en código Python.
- Ejecuta pruebas con tres casos diferentes y registra los resultados.
- Realiza una demostración rápida a otro equipo, que debe intentar encontrar un caso no contemplado.

Diferenciación: Los estudiantes que requieren apoyo utilizan un modelo de código con comentarios guía. Quienes terminan antes agregan una segunda recomendación o una condición combinada con `and` u `or`.

Transición: El docente relaciona las pruebas con la siguiente sesión: “Si el usuario necesita registrar varios datos, repetir el código manualmente no es eficiente; aprenderemos a usar ciclos”.

Fase de Cierre

Tiempo estimado: 10 minutos.

- **Síntesis, 5 minutos:** Cada equipo completa en el tablero la frase: “Una condición sirve para..., se escribe con..., y debe probarse con...”.
- **Reflexión, 3 minutos:** Responden: “¿Qué caso puede hacer fallar mi condición?”, “¿Qué comparación utilicé y por qué?” y “¿Qué aprendí al probar con datos distintos?”.
- **Retroalimentación y transferencia, 2 minutos:** El docente destaca ejemplos de condiciones claras y deja como reto escribir en casa dos decisiones que podría tomar su aplicación.

Sesión 3: Repetir y organizar datos

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Usar ciclos y colecciones de datos para registrar varios elementos sin repetir innecesariamente las instrucciones.

- **Docente, 5 minutos:** Pregunta: “¿Cómo pediríamos diez tareas o diez gastos sin copiar diez veces las mismas líneas?”.
- **Estudiantes, 3 minutos:** Proponen una solución y detectan qué parte del código tendría que repetirse.

- **Docente, 2 minutos:** Realiza una demostración breve de un ciclo que imprime los números del 1 al 5.

Contextualización: Se conecta el concepto con listas de canciones, productos de una compra, tareas pendientes y registros de hábitos. Se enfatiza que los ciclos permiten automatizar acciones repetitivas.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Presentación del contenido

Docente, 15 minutos: Explica las listas como colecciones ordenadas y presenta ejemplos:

```
tareas = ["leer", "investigar", "practicar"]
```

```
for tarea in tareas:
print(tarea)
```

Luego muestra `range()` para repetir una cantidad determinada de veces y el ciclo `while` para repetir mientras se cumpla una condición. Advierte que un `while` debe modificar su condición para evitar un ciclo infinito.

Actividad 1. Ordenar y recorrer colecciones

Objetivo: Crear listas y recorrer sus elementos.

Tiempo: 55 minutos. **Organización:** Parejas.

- Los estudiantes escriben una lista de cinco elementos vinculados con su proyecto.
- Usan `for` para mostrar cada elemento con un número.
- Agregan un elemento mediante `append()` y comprueban el resultado.
- Calculan cuántos elementos hay usando `len()`.
- Explican a otra pareja qué representa cada elemento de su lista.

Docente: Pregunta: “¿Qué variable representa cada elemento?”, “¿Qué ocurre si la lista está vacía?”.

Evidencia: Programa de recorrido de lista y explicación escrita.

Actividad 2. Registro repetido con ciclos

Objetivo: Aplicar ciclos para capturar varios datos.

Tiempo: 75 minutos. **Organización:** Equipos de tres.

- El docente entrega el reto: “Construyan un programa que solicite cinco registros y los guarde en una lista”.
- Los estudiantes deciden qué registrarán: gastos, tareas, minutos de actividad o cantidad de residuos.
- Escriben el ciclo con `for`, convierten los datos numéricos con `int()` o `float()` y muestran la lista final.
- Calculan un total o promedio sencillo.
- Prueban valores normales, cero y un dato inesperado, y anotan qué ocurrió.

Docente: Pregunta: “¿Cuántas veces se repite la solicitud?”, “¿Dónde se guarda cada dato?”, “¿El total se actualiza dentro o fuera del ciclo?”.

Evidencia: Programa de registro múltiple, lista de datos y cálculo.

Actividad 3. Integración de ciclos al proyecto

Objetivo: Incorporar ciclos y listas en la aplicación.

Tiempo: 60 minutos. **Organización:** Equipos.

- El equipo identifica una parte del proyecto que requiere varios registros.
- Reemplaza instrucciones repetidas por una lista y un ciclo.
- Integra el cálculo o recorrido con las condiciones construidas en la sesión anterior.
- Realiza una prueba con cinco datos y presenta el resultado a un equipo vecino.

Diferenciación: Quienes necesitan ayuda utilizan un diagrama del ciclo y un código base. Quienes avanzan agregan un menú que permita elegir cuántos registros desea ingresar el usuario.

Transición: El docente señala: “Ya podemos guardar varios datos, pero el código empieza a crecer. En la siguiente sesión aprenderemos a dividirlo en funciones reutilizables”.

Fase de Cierre

Tiempo estimado: 10 minutos.

- **Síntesis, 5 minutos:** Los estudiantes elaboran un mini mapa con los conceptos `lista`, `for`, `while`, `append()` y `len()`, unidos mediante ejemplos.
- **Reflexión, 3 minutos:** Responden: “¿Qué repetición resolví con un ciclo?”, “¿Qué ventaja tiene guardar datos en una lista?” y “¿Qué prueba debo realizar antes de confiar en el resultado?”.
- **Retroalimentación, 2 minutos:** El docente revisa rápidamente que cada equipo tenga una lista, un ciclo y una evidencia de prueba.

Tarea: Dibujar en papel el menú ideal de la aplicación final con sus opciones principales.

Sesión 4: Funciones y diseño modular

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Organizar el programa en funciones para que sea más claro, reutilizable y fácil de corregir.

- **Docente, 5 minutos:** Muestra un programa largo y pregunta: “¿Qué parte sería difícil de modificar si todo estuviera escrito seguido?”.
- **Estudiantes, 3 minutos:** Identifican bloques posibles: bienvenida, registro, cálculo y recomendación.
- **Docente, 2 minutos:** Presenta la comparación con una cocina: cada función realiza una tarea específica y puede utilizarse cuando se necesite.

Contextualización: Se explica que los programas reales se organizan en partes para que varios programadores puedan trabajar y para que una modificación no afecte todo el sistema.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Presentación del contenido

Docente, 15 minutos: Modela:

```
def saludar(nombre):
return "Hola, " + nombre


mensaje = saludar("Ana")
print(mensaje)
```

Explica nombre de la función, parámetros, instrucciones internas, valor de retorno y llamada. Recalca que una función debe realizar una tarea concreta y que sus nombres deben ser descriptivos.

Actividad 1. Rompecódigo modular

Objetivo: Identificar tareas que pueden convertirse en funciones.

Tiempo: 50 minutos. **Organización:** Equipos.

- El docente entrega un programa dividido en tarjetas desordenadas: entrada, cálculo, decisión y salida.
- Los equipos agrupan las tarjetas en tareas y asignan un nombre a cada función.
- Escriben qué datos recibe cada función y qué resultado debe devolver.
- Comparan su diseño con otro equipo y justifican una diferencia.

Docente: Pregunta: “¿Esta función tiene una sola responsabilidad?”, “¿Qué necesita conocer para trabajar?”.

Evidencia: Diagrama modular con funciones, entradas y salidas.

Actividad 2. Construir funciones

Objetivo: Diseñar funciones con parámetros y valores de retorno.

Tiempo: 75 minutos. **Organización:** Parejas.

- Los estudiantes crean una función de bienvenida, una función de cálculo y una función de recomendación.
- Prueban cada función por separado con datos sencillos.
- Usan `return` para enviar resultados a la parte principal.
- Agregan comentarios breves que expliquen la responsabilidad de cada función.
- Intercambian archivos y otra pareja prueba las funciones con valores distintos.

Docente: Observa si los estudiantes confunden imprimir con devolver. Pregunta: “¿Podrías usar esta función con otro dato?”, “¿Qué cambia si modificas solo el cálculo?”.

Evidencia: Tres funciones probadas y comentadas.

Actividad 3. Refactorización del proyecto

Objetivo: Organizar la aplicación en módulos funcionales.

Tiempo: 65 minutos. **Organización:** Equipos.

- El equipo revisa su código y subraya bloques repetidos o difíciles de leer.
- Convierte al menos tres bloques en funciones.
- Construye un menú principal que llame las funciones según la opción seleccionada.
- Prueba cada opción y registra los errores encontrados.
- Actualiza el diagrama del proyecto para reflejar la nueva estructura.

Diferenciación: Los estudiantes que necesitan apoyo reciben una plantilla con `def`, parámetros y `return` incompletos. Quienes terminan antes incorporan una función para validar entradas o permiten repetir el menú.

Transición: El docente indica: “Una aplicación organizada se puede probar mejor. En la próxima sesión usaremos pruebas sistemáticas y guardaremos información en un archivo”.

Fase de Cierre

Tiempo estimado: 10 minutos.

- **Síntesis, 5 minutos:** Cada estudiante escribe el nombre de una función de su proyecto y completa: “Recibe..., realiza..., devuelve...”.
- **Reflexión, 3 minutos:** Responden: “¿Qué parte del programa quedó más clara al convertirla en función?”, “¿Qué diferencia hay entre `print` y `return`?” y “¿Cómo facilita una función la corrección del programa?”.
- **Retroalimentación, 2 minutos:** El docente revisa nombres de funciones, parámetros y retornos, y comunica los criterios de la prueba de prototipo.

Tarea: Escribir tres casos de prueba para la aplicación: uno esperado, uno límite y uno con dato incorrecto.

Sesión 5: Pruebas, archivos y depuración

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Probar, corregir y mejorar la aplicación, comprendiendo que los errores son parte normal del proceso de programación.

- **Docente, 5 minutos:** Proyecta un error de sintaxis y pregunta: “¿Qué información nos está dando el mensaje de Python?”.
- **Estudiantes, 3 minutos:** En parejas clasifican tres errores como de sintaxis, ejecución o lógica.
- **Docente, 2 minutos:** Presenta el reto: “Hoy cada equipo deberá encontrar y corregir al menos cinco problemas reales o intencionales”.

Contextualización: Se relaciona la depuración con probar una bicicleta o revisar una receta antes de usarla. Una aplicación útil debe producir resultados consistentes y manejar situaciones inesperadas.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Presentación del contenido

Docente, 15 minutos: Explica una estrategia de depuración: leer el error, ubicar la línea, reproducir el problema, formular una hipótesis, cambiar una cosa y volver a probar. Presenta el uso básico de archivos:

```
with open("datos.txt", "w") as archivo:
archivo.write("Registro guardado")
```

También muestra cómo usar `try` y `except` para evitar que el programa se cierre ante una entrada no numérica.

Actividad 1. Detective de errores

Objetivo: Clasificar y corregir errores sencillos.

Tiempo: 55 minutos. **Organización:** Parejas.

- El docente entrega cuatro fragmentos con errores: comilla faltante, variable mal escrita, conversión incorrecta y condición equivocada.
- Los estudiantes ejecutan cada fragmento y copian el mensaje de error.
- Escriben una hipótesis y realizan una corrección.
- Comprueban que el programa funcione después del cambio.

Docente: Evita dar la respuesta inmediata y pregunta: “¿Qué línea señala Python?”, “¿El problema impide ejecutar o produce un resultado incorrecto?”.

Evidencia: Tabla de error, causa, corrección y resultado.

Actividad 2. Guardar y recuperar información

Objetivo: Usar archivos de texto para conservar datos del programa.

Tiempo: 70 minutos. **Organización:** Equipos de tres.

- Los estudiantes crean una función `guardar_dato()` que escriba un registro en un archivo de texto.
- Crean una función `leer_datos()` que muestre el contenido guardado.
- Prueban cerrar y volver a abrir el programa para comprobar que la información permanece.
- Agregan una opción de menú para guardar y otra para consultar.
- Comentan el código indicando qué archivo se utiliza y para qué.

Docente: Verifica el uso de `with open` y pregunta: “¿Qué pasaría si el archivo todavía no existe?”, “¿Qué información conviene guardar?”.

Evidencia: Archivo de datos y funciones de lectura y escritura.

Actividad 3. Prueba cruzada del prototipo

Objetivo: Depurar la aplicación mediante pruebas realizadas por usuarios reales.

Tiempo: 65 minutos. **Organización:** Equipos de cuatro y parejas de prueba.

- Cada equipo entrega su prototipo y una guía de tres tareas a un equipo visitante.
- El equipo visitante ejecuta las tareas sin recibir ayuda durante los primeros cinco minutos.
- Registra dificultades de comprensión, errores y resultados inesperados.
- El equipo creador clasifica los hallazgos por prioridad: urgente, importante o mejora.
- Corrige al menos tres problemas y vuelve a probar.

Diferenciación: Los equipos que requieren apoyo reciben casos de prueba preparados. Quienes avanzan antes agregan mensajes de error claros, confirmación antes de salir y validación de datos vacíos.

Transición: El docente anuncia: “La próxima sesión será de versión final: deberán presentar una aplicación funcional, explicar su código y demostrar que fue probada”.

Fase de Cierre

Tiempo estimado: 10 minutos.

- **Síntesis, 5 minutos:** Cada equipo actualiza su registro de cambios con tres columnas: problema encontrado, solución aplicada y evidencia de prueba.
- **Reflexión, 3 minutos:** Responden: “¿Qué error me enseñó algo importante?”, “¿Cómo comprobé que mi corrección funcionaba?” y “¿Qué entrada inesperada debe manejar mi programa?”.
- **Retroalimentación, 2 minutos:** El docente revisa que cada equipo tenga prototipo ejecutable, archivo de datos y lista de mejoras para la versión final.

Tarea: Preparar una presentación de cinco minutos con problema, solución, demostración y aprendizaje principal.

Sesión 6: Feria de aplicaciones y reflexión final

Fase de Inicio

Tiempo estimado: 10 minutos.

Propósito de la sesión: Finalizar, presentar y evaluar una aplicación de Python, argumentando cómo responde a un problema real.

- **Docente, 5 minutos:** Recuerda los criterios de la rúbrica: funcionamiento, uso de conceptos, claridad del código, pruebas y presentación.
- **Estudiantes, 3 minutos:** Revisan su lista de pendientes y distribuyen los roles: programador de apoyo, demostrador, relator y encargado de evidencias.
- **Docente, 2 minutos:** Explica la dinámica de la feria y la norma: cada visitante debe hacer una pregunta respetuosa y una sugerencia útil.

Contextualización: El docente señala que presentar software implica explicar su utilidad a personas que no conocen el código. Programar también exige comunicar, recibir críticas y mejorar.

Fase de Desarrollo

Tiempo estimado: 220 minutos.

Actividad 1. Versión final y ensayo

Objetivo: Crear una versión estable y documentada de la aplicación.

Tiempo: 65 minutos. **Organización:** Equipos.

- El equipo ejecuta la aplicación desde el inicio y revisa cada opción del menú.
- Comprueba que las variables tengan nombres claros, que el código esté indentado y que existan comentarios breves.
- Ejecuta los casos de prueba y guarda capturas de los resultados.
- Prepara una carpeta con el archivo `.py`, archivos de datos, manual breve y presentación.
- Ensaya una explicación de cinco minutos utilizando la estructura: problema, usuarios, funcionamiento, código y mejoras futuras.

Docente: Usa una lista de cotejo y formula preguntas: “¿Qué ocurre si el usuario escribe una opción inválida?”, “¿Qué función es central?”, “¿Qué evidencia demuestra que funciona?”.

Evidencia: Versión final, manual y ensayo.

Actividad 2. Feria de aplicaciones

Objetivo: Presentar y evaluar colaborativamente una solución programada.

Tiempo: 90 minutos. **Organización:** Estaciones de cuatro estudiantes.

- La mitad de los equipos presenta mientras la otra mitad visita; después intercambian roles.
- Cada equipo dispone de cinco minutos para explicar y tres minutos para responder preguntas.
- Los visitantes completan una ficha con: utilidad, concepto de Python observado, fortaleza y sugerencia.
- El docente observa el funcionamiento, la explicación y la participación de todos.

Preguntas guía para visitantes: “¿Qué problema resuelve?”, “¿Qué datos necesita?”, “¿Qué pasaría con un dato inesperado?” y “¿Qué mejora le agregarías?”.

Evidencia: Demostración pública, ficha de coevaluación y registro docente.

Actividad 3. Mejora posterior a la retroalimentación

Objetivo: Evaluar comentarios y realizar una mejora concreta.

Tiempo: 50 minutos. **Organización:** Equipos.

- El equipo revisa las fichas recibidas y elige una sugerencia viable.
- Explica por escrito por qué la acepta, la adapta o la descarta.
- Realiza una mejora en el código y ejecuta una prueba final.
- Actualiza el archivo de cambios y prepara una frase sobre el aprendizaje más importante.

Diferenciación: Los estudiantes que necesitan apoyo pueden mejorar mensajes, instrucciones o documentación. Quienes avanzan incorporan una nueva función, una validación o una opción de estadísticas, sin comprometer la estabilidad del programa.

Transición: El docente reúne al grupo y afirma: “La feria termina, pero el aprendizaje continúa: ahora vamos a explicar qué aprendimos sobre programar y sobre trabajar como equipo”.

Fase de Cierre

Tiempo estimado: 10 minutos.

Síntesis, reflexión y retroalimentación final

- **Síntesis, 3 minutos:** En plenaria, el docente construye una ruta en el tablero: problema → algoritmo → datos → decisiones → ciclos → funciones → pruebas → solución.
- **Reflexión metacognitiva, 4 minutos:** Cada estudiante responde por escrito: “¿Qué concepto de Python puedo aplicar sin copiar un ejemplo?”, “¿Qué error o dificultad logré superar?”, “¿Cómo contribuyó mi equipo al producto final?” y “¿Qué mejoraría si tuviera dos semanas más?”.
- **Retroalimentación, 2 minutos:** El docente comunica fortalezas generales y una recomendación individual o grupal basada en la rúbrica.
- **Transferencia, 1 minuto:** Se invita a identificar un problema de la comunidad que podría resolverse con una futura aplicación y a conservar el portafolio como evidencia de aprendizaje.

Producto final: Aplicación de consola en Python, código documentado, archivo de datos, manual breve, casos de prueba, presentación y reflexión individual.

Evaluación

Tipo y momentos de evaluación

- **Diagnóstica:** En la Fase de Inicio de la Sesión 1, mediante la pregunta sobre instrucciones para un robot, la nota adhesiva de problemas cotidianos y la observación del manejo inicial del editor.
- **Formativa:** Durante todas las actividades de desarrollo, mediante revisión de pseudocódigos, observación directa, tablas de pruebas, código compartido, preguntas guía y retroalimentación entre equipos.
- **Sumativa:** En la Sesión 6, durante la feria de aplicaciones, la entrega del código final, la demostración y la reflexión individual.

Criterios de evaluación vinculados con los objetivos

- **Analiza el problema y diseña una solución pertinente:** identifica usuarios, entradas, procesos y salidas, y presenta un algoritmo coherente.
- **Aplica conceptos básicos de Python:** utiliza correctamente variables, entradas, salidas, operadores, condicionales, ciclos, listas y funciones.

- **Organiza el programa:** emplea funciones con nombres claros, comentarios pertinentes, estructura legible e interacción comprensible.
- **Depura y prueba:** identifica errores, registra correcciones y demuestra el funcionamiento con casos normales, límites e inesperados.
- **Comunica y trabaja colaborativamente:** presenta el producto, explica decisiones técnicas, escucha retroalimentación y evidencia participación responsable.

Instrumentos sugeridos

- Rúbrica analítica del proyecto con niveles: inicial, básico, satisfactorio y avanzado.
- Lista de cotejo de sintaxis, estructuras de control, funciones, validación y documentación.
- Guía de observación del trabajo colaborativo y la resolución de problemas.
- Portafolio digital con pseudocódigo, capturas, tablas de pruebas, versiones y registro de cambios.
- Coevaluación de la feria de aplicaciones.
- Autoevaluación individual mediante las preguntas metacognitivas de la Sesión 6.

Evidencias de aprendizaje

- Ficha de análisis del problema y pseudocódigo: evidencia del análisis y diseño.
- Programas de práctica con entradas, salidas, condiciones, ciclos y listas: evidencia de aplicación de fundamentos.
- Diagrama modular y código con funciones: evidencia de diseño organizado.
- Tabla de errores y casos de prueba: evidencia de depuración.
- Aplicación final ejecutable, archivo de datos, manual y presentación: evidencia de creación y comunicación de una solución.
- Reflexión individual y coevaluación: evidencia de metacognición y colaboración.

Enriquecimientos

Inicio - Activar

Actividad de activación: “¿Cómo resolverías este problema?”

Duración: 8 minutos

Momento de aplicación: Inicio de la primera sesión

Propósito: Activar los conocimientos previos de los estudiantes sobre instrucciones, secuencias, decisiones y resolución de problemas, relacionándolos con la programación en Python.

Materiales

- Tablero o proyector.

- Una situación problema escrita en el tablero: “Una aplicación debe ayudar a un estudiante a saber si puede participar en una actividad según su edad y el tiempo disponible”.
- Hojas pequeñas o notas adhesivas, una por estudiante o pareja.

Desarrollo de la actividad

Tiempo	Acción docente	Acción de los estudiantes
2 minutos	Presentar la situación: “Imaginen que deben crear una aplicación que reciba la edad de una persona y el tiempo que tiene disponible. La aplicación debe indicar si puede participar en una actividad de 30 minutos”.	Escuchar el reto e identificar qué información necesitaría la aplicación.
3 minutos	Solicitar que, individualmente o en parejas, escriban los pasos que debería seguir la aplicación para responder correctamente.	Organizar la solución como una secuencia de instrucciones, por ejemplo: pedir datos, comparar el tiempo disponible y mostrar un mensaje.
2 minutos	Invitar a dos o tres estudiantes a compartir sus pasos. Guiar la conversación hacia las ideas de entrada, procesamiento, decisión y salida.	Explicar sus propuestas y comparar diferentes formas de resolver el problema.
1 minuto	Relacionar las respuestas con Python: “Programar consiste en convertir una solución organizada en instrucciones que la computadora pueda ejecutar”.	Reconocer que sus pasos pueden transformarse posteriormente en un programa.

Preguntas orientadoras

- ¿Qué datos necesita conocer la aplicación antes de responder?
- ¿Qué tendría que comparar o calcular?
- ¿Qué debería ocurrir si el tiempo disponible es suficiente?
- ¿Qué debería ocurrir si el tiempo no alcanza?
- ¿Por qué es importante organizar las instrucciones en un orden?
- ¿Han utilizado alguna aplicación que tome decisiones a partir de los datos que ingresan?

Conexión con los objetivos de aprendizaje

La actividad permite que los estudiantes comprendan que programar implica analizar un problema, dividirlo en pasos y establecer instrucciones claras. Estas ideas servirán como base para aprender en Python conceptos como variables, entrada de datos, operadores, condicionales y salida de información, que utilizarán posteriormente en el desarrollo de su aplicación para resolver un problema real.

Evidencia rápida de aprendizaje

El docente revisa si los estudiantes incluyen al menos tres elementos en su propuesta: los datos de entrada, una acción o comparación y un resultado o mensaje final. No es necesario que utilicen código durante esta actividad; el propósito es identificar sus ideas iniciales sobre la lógica de programación.