Python Creativo: Introducción a la Programación y al Pensamiento Computacional

Tecnología e Informática | Informática | para estudiantes de secundaria (12-15 años) | 4 semanas

Descripción del Curso

Este curso introduce a los estudiantes de secundaria, entre 12 y 15 años, en los fundamentos de la programación mediante el lenguaje Python. Su propósito es desarrollar el pensamiento computacional, el razonamiento lógico y la capacidad para resolver problemas de la vida cotidiana utilizando instrucciones claras, ordenadas y verificables.

Está dirigido a estudiantes que se inician en la programación, por lo que no requiere experiencia previa. A lo largo de cuatro semanas, se abordarán progresivamente las variables, los tipos de datos, los operadores, las estructuras condicionales, los ciclos y las funciones. Cada tema se relacionará con situaciones cercanas a los estudiantes, como cálculos, juegos sencillos, conversiones, toma de decisiones y organización de información.

El enfoque metodológico será práctico, interactivo y basado en retos. Se combinarán explicaciones breves, demostraciones, ejercicios guiados, trabajo colaborativo, experimentación y pequeños proyectos. Se promoverá que los estudiantes planifiquen sus soluciones, escriban código, prueben sus programas, identifiquen errores y mejoren sus resultados.

Al finalizar, los estudiantes podrán crear programas sencillos en Python, interpretar código básico, utilizar estructuras fundamentales de programación y aplicar estrategias de descomposición de problemas, reconocimiento de patrones y elaboración de algoritmos. También fortalecerán la creatividad, la perseverancia y la confianza para aprender tecnología.

Objetivos Generales

- Identificar los conceptos fundamentales de la programación y del pensamiento computacional, relacionándolos con situaciones de la vida cotidiana.
- Aplicar variables, tipos de datos, operadores y funciones de entrada y salida para construir programas básicos en Python.
- Implementar estructuras condicionales y repetitivas para resolver problemas que requieran decisiones o acciones iterativas.
- Crear y organizar programas sencillos mediante funciones, siguiendo una secuencia lógica y buenas prácticas básicas de programación.
- Probar, explicar y mejorar un programa funcional que resuelva un problema cercano a su contexto.

Competencias

- Analiza problemas cotidianos y los descompone en pasos ordenados para diseñar soluciones computacionales sencillas.
- Utiliza variables, tipos de datos, operadores y expresiones básicas de Python para procesar información.
- Implementa estructuras condicionales y ciclos para controlar el comportamiento de programas sencillos.
- Define y utiliza funciones básicas para organizar, reutilizar y comprender mejor el código.
- Prueba, depura y mejora programas identificando errores de sintaxis, lógica o ejecución.
- Comunica el propósito, el funcionamiento y los resultados de sus programas mediante explicaciones claras y ejemplos.

Requerimientos

- Interés por resolver problemas, crear soluciones y aprender mediante la experimentación.
- Conocimientos básicos de uso del computador: teclado, archivos, carpetas y navegación en aplicaciones.
- Acceso a un computador con conexión a Internet, de manera individual o en parejas.
- Entorno de programación Python instalado o acceso a una plataforma en línea, como Python Turtle, Replit o Google Colab.
- Cuaderno o documento digital para registrar algoritmos, ejemplos, dudas y reflexiones.
- Material de apoyo proporcionado por el docente: guías, ejercicios, retos y rúbricas de evaluación.

Unidades del Curso

Unidad 1: Pensamiento computacional y primeros pasos con Python

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de explicar los conceptos de pensamiento computacional, algoritmo y secuencia mediante ejemplos de situaciones cotidianas y soluciones expresadas paso a paso.
- Al finalizar la unidad, el estudiante será capaz de identificar y describir las funciones básicas del entorno de Python para escribir, ejecutar y revisar programas sencillos.
- Al finalizar la unidad, el estudiante será capaz de elaborar programas básicos en Python que incluyan comentarios y mensajes en pantalla mediante la función `print()`, respetando una secuencia lógica.
- Al finalizar la unidad, el estudiante será capaz de utilizar la función `input()` para solicitar datos al usuario y mostrar respuestas relacionadas con la información ingresada.
- Al finalizar la unidad, el estudiante será capaz de probar y corregir errores sencillos de sintaxis en un programa de Python, explicando los cambios realizados para lograr su ejecución.

Contenidos Temáticos

1. Pensamiento computacional

Se abordará el pensamiento computacional como una forma ordenada de analizar problemas y diseñar soluciones que puedan ser comprendidas por una persona o ejecutadas por una computadora.

- **Definición:** proceso de analizar un problema, organizar la información y proponer una solución paso a paso.
- **Descomposición:** dividir un problema grande en partes más pequeñas y fáciles de resolver. Ejemplo: preparar una exposición implica investigar, seleccionar información, crear diapositivas y ensayar.
- **Reconocimiento de patrones:** identificar semejanzas o repeticiones en problemas y situaciones.
- **Abstracción:** concentrarse en la información importante y dejar de lado los detalles que no son necesarios.
- **Diseño de algoritmos:** organizar instrucciones ordenadas para resolver un problema.
- **Aplicación cotidiana:** seguir una receta, organizar una mochila o indicar cómo llegar a un lugar son ejemplos de soluciones paso a paso.

2. Algoritmos y secuencias

Se explicará que un algoritmo es un conjunto finito y ordenado de instrucciones para alcanzar un objetivo. La secuencia indica el orden en que deben realizarse las acciones.

- **Características de un algoritmo:** tiene un objetivo, utiliza instrucciones claras, sigue un orden, debe terminar y produce un resultado.
- **Instrucciones precisas:** comparar “prepara una bebida” con “coloca agua en un vaso y agrega una cucharada de polvo”.
- **Secuencia:** cambiar el orden de los pasos puede modificar o impedir el resultado.
- **Entrada, proceso y salida:**
  - **Entrada:** datos o elementos que se necesitan.
  - **Proceso:** acciones que se realizan con esos datos.
  - **Salida:** resultado obtenido.
- **Representación de algoritmos:** descripción con lenguaje cotidiano, lista numerada, pseudocódigo y programa.
- **Ejemplo de algoritmo cotidiano:**
  1. Leer el enunciado de una tarea.
  2. Identificar los materiales necesarios.
  3. Resolver cada punto.
  4. Revisar las respuestas.
  5. Entregar la tarea.

3. Introducción al entorno de Python

Se reconocerán las partes básicas de un entorno de programación y el procedimiento para escribir, guardar, ejecutar y revisar un programa.

- **Python:** lenguaje de programación que permite dar instrucciones a una computadora de manera relativamente sencilla.
- **Entorno de desarrollo:** espacio donde se escribe y ejecuta el código. Puede utilizarse una instalación de Python con IDLE o un entorno web como Replit, Programiz o similar.
- **Editor:** área donde se escriben y modifican las instrucciones.
- **Consola o shell:** espacio donde se ejecutan instrucciones y se observan resultados o mensajes de error.
- **Ejecutar:** ordenar a Python que interprete las instrucciones del programa.
- **Guardar:** almacenar el archivo con extensión `.py`, por ejemplo, `saludo.py`.
- **Revisar:** observar la salida, identificar errores y comparar el resultado con lo esperado.
- **Normas básicas de trabajo:** guardar versiones, escribir nombres descriptivos para los archivos, probar frecuentemente y leer los mensajes de error sin borrar el programa completo.

4. Comentarios y mensajes en pantalla con `print()`

Se elaborarán programas sencillos que muestren información en pantalla y contengan comentarios que expliquen su propósito.

- **Comentarios:** notas para las personas que leen el programa. Python no las ejecuta.
- **Sintaxis del comentario:** se utiliza el símbolo `#`.
- **Mensaje en pantalla:** la función `print()` muestra texto o valores.
- **Cadenas de texto:** los mensajes se escriben entre comillas simples o dobles.
- **Varios mensajes:** pueden utilizarse varias instrucciones `print()` en secuencia.
- **Ejemplo:**

```
# Programa de presentación
print("Hola, mundo")
print("Estoy aprendiendo Python")
```

- **Uso de variables como anticipación:**

```
nombre = "Ana"
print("Hola,", nombre)
```

- **Errores frecuentes:** olvidar comillas, escribir incorrectamente `print`, cerrar de forma incompleta los paréntesis o usar caracteres innecesarios.

5. Entrada de datos con `input()`

Se aprenderá a solicitar información al usuario y utilizarla para generar respuestas personalizadas.

- **Función `input()`:** muestra una pregunta y espera que el usuario escriba una respuesta.
- **Asignación:** la respuesta se almacena en una variable para utilizarla posteriormente.
- **Ejemplo básico:**

```
nombre = input("¿Cómo te llamas? ")
print("Hola,", nombre)
```

- **Secuencia entrada-salida:** primero se solicita el dato y después se muestra una respuesta relacionada.
- **Información textual:** inicialmente, la respuesta de `input()` se trabaja como texto.
- **Conversión básica a número:** cuando se requiera una edad u otro número, puede utilizarse `int()`.

```
edad = int(input("¿Cuántos años tienes? "))
print("El próximo año tendrás", edad + 1)
```

- **Buenas prácticas:** formular preguntas claras, utilizar nombres de variables comprensibles y comprobar que el usuario introduzca el tipo de dato esperado.

6. Prueba y corrección de errores sencillos

Se desarrollará una actitud de prueba, observación y mejora para corregir errores básicos de sintaxis y explicar los cambios realizados.

- **Error de sintaxis:** ocurre cuando las instrucciones no están escritas de acuerdo con las reglas de Python.
- **Errores habituales:**
  - Escribir `Print` en lugar de `print`.
  - Olvidar un paréntesis: `print("Hola"`.
  - Olvidar una comilla: `print("Hola)`.
  - Usar una variable con un nombre diferente al que fue creado.
  - Convertir de manera incorrecta una entrada a número.
- **Proceso de depuración:** ejecutar, observar el mensaje, localizar la línea señalada, formular una hipótesis, cambiar una sola cosa y volver a ejecutar.
- **Registro de cambios:** anotar qué error se encontró, qué modificación se realizó y cuál fue el resultado.
- **Ejemplo de corrección:**

```
# Código con error
print("Mi color favorito es el azul"

# Código corregido
print("Mi color favorito es el azul")
```

- **Importancia del mensaje de error:** no debe considerarse un fracaso, sino una pista para comprender qué necesita corregirse.

7. Integración: programa interactivo sencillo

Se integrarán comentarios, secuencia, `print()` e `input()` en un programa relacionado con una situación cercana a los estudiantes.

- Definir el propósito del programa.
- Identificar los datos que se solicitarán.

- Organizar la secuencia de instrucciones.
- Escribir comentarios breves.
- Solicitar datos con `input()`.
- Mostrar una respuesta personalizada con `print()`.
- Probar el programa con diferentes respuestas.
- Corregir errores y explicar los cambios.

Actividades

Actividad 1. “Instrucciones para un robot humano”

Objetivo: contribuir a la comprensión del pensamiento computacional, los algoritmos, la descomposición y la secuencia.

Descripción paso a paso:

1. El docente explica que una computadora necesita instrucciones claras y ordenadas.
2. Se forman parejas y cada pareja elige una tarea cotidiana: lavarse los dientes, preparar un sándwich, organizar una mochila o enviar un mensaje.
3. Un estudiante escribe entre 6 y 10 instrucciones concretas. El otro estudiante actúa como “robot” y solo puede realizar exactamente lo que está escrito.
4. La pareja prueba el algoritmo. Si una instrucción es ambigua o falta un paso, el “robot” se detiene y solicita aclaración.
5. La pareja revisa el orden y mejora las instrucciones.
6. Cada pareja comparte su algoritmo y explica qué ocurriría si se cambia el orden de dos pasos.

Organización: parejas.

Producto esperado: algoritmo cotidiano escrito en pasos numerados y una explicación breve de la importancia de la secuencia.

Duración estimada: 45 minutos.

Actividad 2. “Mi primer programa: mensajes con propósito”

Objetivo: identificar las funciones básicas del entorno de Python y elaborar un programa con comentarios, `print()` y una secuencia lógica.

Descripción paso a paso:

1. El docente presenta el editor, la consola, el botón u opción para ejecutar y el procedimiento para guardar un archivo.
2. Los estudiantes crean un archivo llamado `presentacion.py`.
3. Escriben un comentario que explique el propósito del programa.

4. Escriben tres o cuatro instrucciones `print()` para mostrar un saludo, su nombre, una afición y una meta de aprendizaje.
5. Guardan y ejecutan el programa.
6. Comparan la salida con el código y verifican si los mensajes aparecen en el orden esperado.
7. Modifican uno de los mensajes y vuelven a ejecutar para observar el cambio.
8. Comparten con un compañero dónde se escribe el código y dónde aparece el resultado.

Ejemplo de referencia:

```
# Programa de presentación personal
print("Hola, soy Alex")
print("Me gusta dibujar")
print("Quiero aprender a crear programas")
```

Organización: individual, con revisión en parejas.

Producto esperado: archivo `.py` ejecutable con al menos un comentario y tres mensajes ordenados.

Duración estimada: 60 minutos.

Actividad 3. “Entrevista interactiva”

Objetivo: utilizar `input()` para solicitar datos y mostrar respuestas relacionadas con la información ingresada.

Descripción paso a paso:

1. El docente demuestra cómo almacenar una respuesta en una variable.
2. Los estudiantes diseñan tres preguntas para una entrevista breve: nombre, pasatiempo, comida favorita, lugar preferido o meta personal.
3. Escriben un programa que solicite las respuestas con `input()`.
4. Utilizan `print()` para crear una respuesta personalizada.
5. Prueban el programa con sus propios datos.
6. Intercambian los programas con un compañero, quien los ejecuta utilizando respuestas diferentes.
7. Revisan si las preguntas son claras y si las respuestas aparecen correctamente.

Ejemplo de referencia:

```
# Entrevista interactiva
nombre = input("¿Cuál es tu nombre? ")
pasatiempo = input("¿Qué actividad disfrutas? ")
lugar = input("¿Cuál es tu lugar favorito? ")

print("Hola,", nombre)
print("Te gusta", pasatiempo)
print("Tu lugar favorito es", lugar)
```

Organización: individual y parejas.

Producto esperado: programa interactivo con al menos tres entradas y tres respuestas relacionadas.

Duración estimada: 75 minutos.

Actividad 4. “Detectives de errores”

Objetivo: probar y corregir errores sencillos de sintaxis, explicando los cambios realizados.

Descripción paso a paso:

1. El docente entrega o proyecta varios fragmentos de código con errores sencillos.
2. En equipos, los estudiantes predicen qué error contiene cada fragmento antes de ejecutarlo.
3. Ejecutan el código y observan el mensaje que presenta Python.
4. Localizan la línea señalada y proponen una corrección.
5. Modifican el código, lo vuelven a ejecutar y comprueban el resultado.
6. Registran cada caso en una tabla con las columnas: código original, error observado, corrección y resultado.
7. Cada equipo explica un caso al grupo utilizando la expresión: “El error estaba en…, lo corregimos cambiando…, y el programa ahora…”

Ejemplos de código para analizar:

```
Print("Hola")
```

```
print("Bienvenido)
```

```
nombre = input("¿Cómo te llamas? "
print("Hola", nombre)
```

Organización: grupos de tres o cuatro estudiantes.

Producto esperado: tabla de depuración con al menos cuatro errores corregidos y explicación de cada cambio.

Duración estimada: 60 minutos.

Actividad 5. Proyecto integrador: “Mi asistente de bienvenida”

Objetivo: integrar pensamiento computacional, secuencia, comentarios, `print()`, `input()`, pruebas y corrección de errores en un programa básico.

Descripción paso a paso:

1. Cada estudiante elige un tema: asistente para una biblioteca, recomendador de actividad, presentación de un club o guía de bienvenida escolar.
2. Escribe primero el algoritmo en lenguaje cotidiano, utilizando entre 6 y 10 pasos.
3. Identifica qué datos debe ingresar el usuario y qué respuesta debe recibir.
4. Convierte el algoritmo en un programa de Python.
5. Incluye al menos dos comentarios, tres instrucciones `print()` y dos instrucciones `input()`.
6. Ejecuta el programa con tres conjuntos de respuestas diferentes.
7. Corrige los errores encontrados y registra al menos un cambio realizado.
8. Presenta el programa a un compañero, quien explica la secuencia que observó.

Organización: individual, con prueba entre compañeros.

Producto esperado: algoritmo escrito, archivo `.py` ejecutable y breve explicación de las pruebas y correcciones realizadas.

Duración estimada: 90 minutos.

Evaluación

Evaluación diagnóstica

Momento: inicio de la unidad.

Qué se evalúa: ideas previas sobre algoritmos, secuencias, instrucciones, programación y experiencias anteriores con Python o herramientas digitales.

Cómo se evalúa:

- Presentar una situación cotidiana, como preparar una merienda, y pedir a los estudiantes que escriban los pasos necesarios.
- Solicitar que ordenen tarjetas con instrucciones desorganizadas.
- Realizar preguntas breves: “¿Qué es un algoritmo?”, “¿Qué crees que hace un programa?”, “¿Dónde se observa el resultado de un código?”.
- Aplicar una actividad corta de reconocimiento de símbolos como `#`, `print()` e `input()`, sin calificación numérica.

Instrumento sugerido: cuestionario diagnóstico de 5 preguntas y lista de cotejo de participación.

Uso de resultados: identificar estudiantes que requieren una introducción adicional al entorno, formar parejas de apoyo y ajustar el nivel de los ejemplos.

Evaluación formativa

Momento: durante toda la unidad.

Qué se evalúa: comprensión progresiva del pensamiento computacional, elaboración de algoritmos, manejo del entorno, uso correcto de `print()` e `input()`, capacidad de probar y corregir.

Cómo se evalúa:

- Observar la participación durante la actividad del “robot humano”.
- Revisar los algoritmos y preguntar qué ocurriría si se altera el orden de los pasos.
- Comprobar que cada estudiante puede crear, guardar y ejecutar un archivo.
- Realizar pausas de revisión durante la escritura de programas.
- Solicitar que expliquen oralmente qué hace cada línea de su código.
- Revisar la tabla de errores y correcciones.
- Utilizar preguntas de salida, por ejemplo: “¿Qué diferencia hay entre `print()` e `input()`?” y “¿Qué hiciste cuando apareció un error?”.

Instrumentos sugeridos: lista de cotejo, guía de observación, revisión de archivos, preguntas orales y diario breve de depuración.

Lista de cotejo formativa:

- Describe una situación mediante pasos ordenados.
- Identifica entrada, proceso y salida en un problema sencillo.
- Ubica el editor y la consola del entorno.
- Guarda el archivo con extensión `.py`.
- Utiliza comentarios con `#`.
- Escribe correctamente `print()`.
- Solicita datos con `input()` y los almacena en variables.
- Prueba el programa y explica al menos una corrección.

Evaluación sumativa

Momento: cierre de la unidad.

Qué se evalúa: logro integrado de los cinco objetivos de la unidad mediante el proyecto “Mi asistente de bienvenida”.

Cómo se evalúa: cada estudiante entrega y presenta:

- Un algoritmo escrito en pasos claros y ordenados.
- Un programa funcional guardado como archivo `.py`.
- Al menos dos comentarios pertinentes.
- Mensajes en pantalla con `print()`.
- Datos solicitados mediante `input()`.
- Respuestas relacionadas con la información ingresada.
- Registro de pruebas realizadas y errores corregidos.
- Una explicación oral o escrita de la secuencia del programa.

Instrumento sugerido: rúbrica analítica de 20 puntos.

- **Pensamiento computacional y algoritmo, 4 puntos:** descompone la situación, identifica los datos y presenta pasos claros, ordenados y finitos.
- **Manejo del entorno, 4 puntos:** crea, guarda, ejecuta y revisa correctamente el programa.
- **Comentarios y `print()`, 4 puntos:** incluye comentarios útiles y mensajes claros en una secuencia lógica.
- **`input()` y respuestas, 4 puntos:** solicita datos de forma comprensible y muestra respuestas relacionadas con ellos.
- **Prueba, corrección y explicación, 4 puntos:** prueba el código, corrige errores sencillos y explica qué cambios realizó.

Escala de desempeño sugerida:

- **16-20 puntos:** logro destacado; el programa funciona y la explicación es clara y completa.
- **11-15 puntos:** logro esperado; cumple la mayoría de los requisitos con errores menores.
- **6-10 puntos:** en proceso; necesita apoyo para organizar la secuencia o corregir errores.

- **0-5 puntos:** requiere refuerzo; presenta dificultades para construir y ejecutar el programa.

Unidad 2: Variables, datos y operadores

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de declarar y utilizar variables para almacenar cadenas de texto, números enteros, números decimales y valores booleanos en programas sencillos de Python.
- Al finalizar la unidad, el estudiante será capaz de identificar y explicar el tipo de dato de diferentes valores, utilizando conversiones de tipo para procesar correctamente datos ingresados por el usuario.
- Al finalizar la unidad, el estudiante será capaz de aplicar operadores aritméticos para realizar cálculos y resolver situaciones prácticas, comprobando que los resultados sean coherentes.
- Al finalizar la unidad, el estudiante será capaz de utilizar operadores relacionales y lógicos para comparar valores y construir expresiones que representen condiciones de situaciones cotidianas.
- Al finalizar la unidad, el estudiante será capaz de diseñar y probar un programa básico que combine variables, tipos de datos y operadores para resolver un problema contextualizado, explicando el propósito de sus instrucciones.

Contenidos Temáticos

1. Introducción a las variables y los datos

Se introducirá la idea de variable como un nombre que permite guardar y reutilizar información en un programa. Se relacionará con situaciones cotidianas, como guardar el nombre de una persona, su edad o su calificación.

- **¿Qué es una variable?** Espacio con nombre que almacena un valor que puede utilizarse durante la ejecución de un programa.
- **Declaración y asignación:** En Python se crea una variable asignando un valor con el signo igual.
- **Ejemplos:**

```
nombre = "Lucía"
edad = 13
altura = 1.58
tiene_permiso = True
```

- **Reglas para nombrar variables:**
  - Usar letras, números y guion bajo.
  - No comenzar el nombre con un número.
  - No utilizar espacios; se puede usar guion bajo.
  - No utilizar palabras reservadas de Python.
  - Elegir nombres claros y relacionados con la información almacenada.
- **Convenciones de nombres:** Se recomienda utilizar minúsculas y guion bajo, por ejemplo, `nombre_completo` o `total_compra`.

- **Actualización del valor:**

```
puntos = 10
puntos = puntos + 5
print(puntos)
```

- **Uso de variables en mensajes:**

```
nombre = "Diego"
print("Hola,", nombre)
```

- **Errores frecuentes:** Diferenciar entre mayúsculas y minúsculas, utilizar nombres poco descriptivos y confundir el nombre de la variable con el texto que contiene.

2. Tipos de datos básicos en Python

Se estudiarán los principales tipos de datos que se utilizarán en programas sencillos y la forma de reconocerlos.

- **Cadenas de texto o `str`:** Representan palabras, frases o caracteres. Se escriben entre comillas simples o dobles.
- **Números enteros o `int`:** Representan números sin parte decimal, como cantidades, edades o puntos.
- **Números decimales o `float`:** Representan valores con parte decimal, como precios, pesos o alturas.
- **Valores booleanos o `bool`:** Solo pueden ser `True` o `False`. Se utilizan para representar condiciones.
- **Identificación del tipo con `type()`:**

```
dato = 25
print(type(dato))
```

- **Diferencia entre texto y número:**

```
edad_texto = "13"  # cadena
edad_numero = 13   # entero
```

- **Importancia del tipo de dato:** El tipo determina qué operaciones pueden realizarse correctamente.
- **Conversión de tipos:**
  - `int()`: convierte a entero.
  - `float()`: convierte a decimal.
  - `str()`: convierte a cadena de texto.
  - `bool()`: convierte a valor booleano en situaciones específicas.
- **Ejemplo de conversiones:**

```
edad = int("14")
precio = float("12.50")
mensaje = str(2025)
```

- **Errores de conversión:** Analizar por qué `int("hola")` produce un error y por qué algunos decimales no pueden convertirse directamente a entero sin perder información.

3. Entrada y salida de información

Se aprenderá a mostrar resultados con `print()` y a recibir datos del usuario con `input()`.

- **Salida de información:**

```
print("Bienvenidos a Python")
print("Resultado:", 8 + 4)
```

- **Entrada de texto:**

```
nombre = input("Escribe tu nombre: ")
print("Hola,", nombre)
```

- **Característica importante de `input()`:** Siempre devuelve una cadena de texto, aunque el usuario escriba un número.
- **Conversión de datos ingresados:**

```
edad = int(input("¿Cuántos años tienes? "))
altura = float(input("¿Cuánto mides? "))
```

- **Mensajes claros:** Las preguntas deben indicar qué dato se espera y, cuando sea necesario, incluir la unidad de medida.
- **Comprobación de datos:** Verificar si el programa calcula resultados coherentes con la información ingresada.

4. Operadores aritméticos

Se aplicarán operadores matemáticos para resolver cálculos y problemas cotidianos.

- **Suma:** `+`
- **Resta:** `-`
- **Multiplicación:** `*`
- **División:** `/`, cuyo resultado normalmente es decimal.
- **División entera:** `//`, que obtiene la parte entera del resultado.
- **Residuo o módulo:** `%`, que obtiene lo que sobra de una división.
- **Potencia:** `**`.
- **Orden de las operaciones:** Primero se resuelven los paréntesis, luego potencias, multiplicaciones, divisiones y finalmente sumas y restas.
- **Uso de paréntesis:** Permite indicar claramente qué operación debe realizarse primero.
- **Ejemplos:**

```
total = 4 * 3
promedio = (8 + 7 + 9) / 3
sobrante = 17 % 5
cuadrado = 6 ** 2
```

- **Situaciones prácticas:** Calcular precios, promedios, distancias, áreas, cantidades de materiales y distribución de objetos.
- **Comprobación de coherencia:** Revisar unidades, rangos y resultados esperados antes de aceptar un cálculo.

5. Operadores relacionales

Se utilizarán comparaciones para construir expresiones que producen valores booleanos: `True` o `False`.

- **Igual que:** ==
- **Diferente de:** `!=`
- **Mayor que:** >
- **Menor que:**
- **Mayor o igual que:** >=
- **Menor o igual que:** =
- **Diferencia entre = y ==:** Un signo asigna un valor; dos signos comparan valores.
- **Ejemplos:**

```
edad = 14
es_mayor = edad >= 18
print(es_mayor)
```

- **Comparación de textos:** Se pueden comparar cadenas, teniendo en cuenta que las mayúsculas y minúsculas pueden influir en el resultado.

6. Operadores lógicos

Se combinarán condiciones para representar situaciones que requieren más de una comparación.

- **`and`:** El resultado es verdadero cuando todas las condiciones son verdaderas.
- **`or`:** El resultado es verdadero cuando al menos una condición es verdadera.
- **`not`:** Invierte el valor lógico de una condición.
- **Ejemplos:**

```
edad = 14
tiene_autorizacion = True

puede_participar = edad >= 12 and tiene_autorizacion
print(puede_participar)
```

- **Construcción de condiciones cotidianas:** Acceso a una actividad, cumplimiento de requisitos, clasificación de resultados o elección de una opción.
- **Tablas de verdad sencillas:** Representar combinaciones de valores `True` y `False` para comprender `and`, `or` y `not`.
- **Uso de paréntesis:** Agrupar condiciones complejas para hacerlas más claras y evitar interpretaciones incorrectas.

7. Integración de variables, tipos y operadores

Se integrarán los contenidos mediante un programa básico que reciba datos, los convierta, realice cálculos y evalúe condiciones.

- **Identificación del problema:** Determinar qué se necesita calcular o decidir.
- **Datos de entrada:** Definir qué información ingresará el usuario.

- **Variables:** Elegir nombres claros para almacenar los datos.
- **Tipos de datos:** Determinar si cada dato es texto, entero, decimal o booleano.
- **Procesamiento:** Aplicar conversiones y operadores.
- **Salida:** Mostrar resultados y mensajes comprensibles.
- **Pruebas:** Probar datos normales, valores límite y casos que podrían generar errores.
- **Explicación del programa:** Describir el propósito de las instrucciones principales.
- **Ejemplo integrador: cálculo de promedio:**

```
nombre = input("Nombre del estudiante: ")
nota1 = float(input("Primera nota: "))
nota2 = float(input("Segunda nota: "))
nota3 = float(input("Tercera nota: "))

promedio = (nota1 + nota2 + nota3) / 3
aprobado = promedio >= 6

print("Estudiante:", nombre)
print("Promedio:", promedio)
print("¿Aprobó?", aprobado)
```

8. Buenas prácticas y depuración

Se promoverá la escritura de programas claros, ordenados y fáciles de revisar.

- Usar nombres de variables descriptivos.
- Separar la entrada, el procesamiento y la salida.
- Agregar comentarios breves cuando una instrucción necesite explicación.
- Probar el programa con diferentes datos.
- Leer los mensajes de error sin borrar automáticamente el código.
- Revisar los paréntesis, las comillas, la indentación y los nombres de variables.
- Comparar el resultado del programa con un cálculo realizado manualmente.

Actividades

Actividad 1. Clasificadores humanos de datos

Objetivo: Identificar y explicar los tipos de datos `str`, `int`, `float` y `bool`.

Descripción paso a paso:

- El docente prepara tarjetas con valores como `"bicicleta"`, `25`, `3.5`, `True`, `"12"` y `False`.
- Los estudiantes clasifican las tarjetas en cuatro grupos: texto, entero, decimal y booleano.
- Se discuten los casos que pueden causar confusión, especialmente `12` y `"12"`.
- Cada pareja escribe una variable adecuada para tres tarjetas, por ejemplo: `edad = 12`.
- En Python, comprueban algunos ejemplos usando `type()`.

- El grupo explica por qué cada valor pertenece a un tipo determinado.

Organización: Parejas y puesta en común.

Producto esperado: Tabla de clasificación con el valor, el tipo de dato y un ejemplo de variable.

Duración estimada: 45 minutos.

Actividad 2. Mi ficha personal en Python

Objetivo: Declarar y utilizar variables con cadenas de texto, números enteros, números decimales y valores booleanos.

Descripción paso a paso:

- Cada estudiante diseña una ficha personal ficticia o utiliza datos que prefiera compartir, evitando información sensible.
- Debe crear variables para nombre, edad, altura aproximada, color favorito y una condición booleana, como `le_gusta_programar`.
- Escribe un programa que muestre todos los datos con mensajes claros.
- Utiliza `type()` para verificar al menos cuatro tipos de datos.
- Cambia uno de los valores y observa cómo se modifica la salida.
- Revisa los nombres de las variables y corrige aquellos que no sean descriptivos o que produzcan errores.

Organización: Individual, con revisión entre parejas.

Producto esperado: Programa funcional y una breve explicación del tipo de dato utilizado en cada variable.

Duración estimada: 60 minutos.

Actividad 3. Calculadora de situaciones cotidianas

Objetivo: Aplicar operadores aritméticos y conversiones de tipo para resolver una situación práctica y comprobar la coherencia de los resultados.

Descripción paso a paso:

- El docente presenta varias situaciones: calcular el costo de una merienda, dividir una cuenta, calcular el promedio de notas o determinar el área de una superficie.
- Cada pareja elige una situación y define los datos que el usuario debe ingresar.
- Elabora una lista de variables, indicando el tipo de cada una.
- Programa las entradas con `input()` y realiza las conversiones necesarias con `int()` o `float()`.
- Aplica operadores aritméticos y muestra el resultado con una explicación.
- Comprueba el resultado manualmente con un ejemplo sencillo.
- Prueba el programa con al menos tres conjuntos de datos y registra los resultados.

Organización: Parejas.

Producto esperado: Calculadora sencilla, tabla de pruebas y explicación del cálculo utilizado.

Duración estimada: 90 minutos.

Actividad 4. Detectives de condiciones

Objetivo: Utilizar operadores relacionales y lógicos para construir expresiones que representen condiciones cotidianas.

Descripción paso a paso:

- El docente entrega situaciones como: “puede participar en una actividad”, “recibe un descuento” o “puede acceder a un juego”.
- Los estudiantes identifican las condiciones necesarias en cada situación.
- Traducen cada condición a una comparación, por ejemplo, `edad >= 12`.
- Combinan las comparaciones con `and`, `or` o `not`.
- Predicen en una tabla si el resultado será `True` o `False`.
- Prueban las expresiones en Python y comparan los resultados con sus predicciones.
- Modifican los valores de entrada para encontrar casos en los que cambie la respuesta.

Organización: Grupos de tres o cuatro estudiantes.

Producto esperado: Tabla de condiciones, expresiones lógicas y programa de comprobación.

Duración estimada: 60 minutos.

Actividad 5. Reto final: programa solucionador

Objetivo: Diseñar, probar y explicar un programa que combine variables, tipos de datos, conversiones y operadores para resolver un problema contextualizado.

Descripción paso a paso:

- Los estudiantes seleccionan un problema cercano, por ejemplo: presupuesto para una salida, cálculo de consumo de agua, promedio de calificaciones, sistema de puntos o planificador de materiales.
- Describen el problema en pocas líneas e identifican los datos de entrada.
- Elaboran un esquema con entrada, procesamiento y salida.
- Definen las variables y sus tipos de datos.
- Escriben el programa utilizando `input()`, conversiones, operadores aritméticos, relacionales y, si corresponde, lógicos.
- Incluyen mensajes comprensibles para el usuario.
- Realizan al menos tres pruebas: una situación habitual, una situación límite y una situación diferente.
- Corrigen los errores encontrados.
- Presentan el programa y explican el propósito de sus instrucciones principales.

Organización: Individual o parejas.

Producto esperado: Programa funcional, ficha de diseño, tabla de pruebas y presentación breve.

Duración estimada: 120 minutos.

Evaluación

Evaluación diagnóstica

Propósito: Identificar los conocimientos previos sobre variables, tipos de datos, operaciones matemáticas y condiciones.

Qué se evalúa:

- Reconocimiento de números, textos y valores verdadero/falso.
- Comprensión inicial de una variable como información almacenada.
- Resolución de operaciones aritméticas sencillas.
- Interpretación de expresiones como “mayor que” o “igual a”.
- Experiencia previa con Python u otros lenguajes.

Cómo se evalúa: Cuestionario breve de cinco a ocho preguntas y actividad de clasificación de valores. Puede incluir preguntas como: “¿Qué tipo de dato es `"20"`?” o “¿Qué resultado produce `7 > 3`?”.

Instrumento sugerido: Cuestionario diagnóstico y lista de cotejo para registrar dificultades iniciales. No debe utilizarse para asignar una calificación definitiva.

Evaluación formativa

Propósito: Acompañar el aprendizaje y detectar errores mientras los estudiantes programan.

Qué se evalúa:

- Uso de nombres claros y asignación correcta de variables.
- Identificación de los tipos `str`, `int`, `float` y `bool`.
- Conversión adecuada de datos recibidos mediante `input()`.
- Aplicación correcta de operadores aritméticos.
- Diferenciación entre asignación y comparación.
- Construcción de condiciones con operadores relacionales y lógicos.
- Capacidad para probar, revisar y corregir un programa.

Cómo se evalúa:

- Observación durante las actividades prácticas.
- Preguntas de comprobación antes de ejecutar el código.
- Revisión de tablas de predicción y resultados.
- Retroalimentación entre pares.
- Revisión de ejercicios y pequeños retos de programación.
- Registro de errores frecuentes y estrategias utilizadas para solucionarlos.

Instrumentos sugeridos: Lista de cotejo, rúbrica breve de ejercicios, diario de aprendizaje y preguntas de salida.

Ejemplo de pregunta de salida: “Explica por qué es necesario usar `int()` o `float()` cuando se realizan cálculos con datos ingresados por el usuario”.

Evaluación sumativa

Propósito: Valorar el dominio integrado de los objetivos de la unidad mediante un programa contextualizado.

Qué se evalúa:

- Declara y utiliza variables de manera correcta.
- Emplea cadenas, enteros, decimales y booleanos cuando corresponde.
- Identifica y explica los tipos de datos utilizados.
- Convierte correctamente los datos ingresados por el usuario.
- Aplica operadores aritméticos para resolver el problema.
- Utiliza operadores relacionales y lógicos cuando la situación requiere tomar una decisión.
- Comprueba la coherencia de los resultados mediante pruebas.
- Explica el propósito de las instrucciones principales.
- Presenta un código ordenado, comprensible y funcional.

Cómo se evalúa: Proyecto final individual o en parejas, acompañado de una breve presentación o demostración. El estudiante debe mostrar el programa, explicar sus variables y ejecutar los casos de prueba.

Instrumento sugerido: Rúbrica analítica de 20 puntos:

- **Variables y tipos de datos:** 4 puntos.
- **Entrada y conversión de datos:** 4 puntos.
- **Operadores y resolución del problema:** 4 puntos.
- **Pruebas y coherencia de resultados:** 3 puntos.
- **Explicación del código:** 3 puntos.
- **Orden, claridad y colaboración:** 2 puntos.

Condición de logro sugerida: Alcanzar al menos 14 de 20 puntos y demostrar que el programa funciona con los casos de prueba establecidos.

Unidad 3: Decisiones y ciclos

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de explicar el funcionamiento de las estructuras `if`, `elif` y `else` mediante ejemplos relacionados con situaciones cotidianas.
- Al finalizar la unidad, el estudiante será capaz de aplicar estructuras condicionales en programas de Python que tomen decisiones a partir de datos ingresados por el usuario, obteniendo resultados correctos en al menos tres casos de prueba.

- Al finalizar la unidad, el estudiante será capaz de utilizar ciclos `for` y `while` para repetir instrucciones y resolver problemas sencillos, evitando repeticiones innecesarias del código.
- Al finalizar la unidad, el estudiante será capaz de construir y probar un programa interactivo que combine condicionales y ciclos para resolver un problema cercano a su contexto, explicando su funcionamiento y corrigiendo al menos un error detectado.

Contenidos Temáticos

1. El control del flujo en un programa

- **Descripción:** Se estudiará cómo un programa puede seguir diferentes caminos según los datos recibidos o las condiciones que se cumplan.
- **Secuencia y decisiones:** Una instrucción puede ejecutarse siempre, solo en ciertas condiciones o repetirse varias veces.
- **Situaciones cotidianas:** Elegir ropa según el clima, decidir si se puede participar en una actividad según la edad o clasificar una calificación.
- **Palabras clave:** `if`, `elif`, `else`, `for` y `while`.

2. Valores, variables y operadores para tomar decisiones

- **Variables:** Espacios donde se almacenan datos, por ejemplo, `edad`, `temperatura` o `puntaje`.
- **Entrada de datos:** Uso de `input()` para recibir información del usuario.
- **Conversión de tipos:** Conversión de texto a número mediante `int()` y `float()`.
- **Operadores de comparación:** ==, `!=`, >, , >= y =.
- **Operadores lógicos:** `and`, `or` y `not` para combinar o negar condiciones.
- **Valores booleanos:** Diferencia entre `True` y `False`.
- **Error frecuente:** Diferenciar entre =, que asigna un valor, y ==, que compara dos valores.

3. Estructura condicional `if`

- **Descripción:** Permite ejecutar un bloque de instrucciones únicamente cuando una condición es verdadera.
- **Sintaxis básica:**

```
if condicion:
    instruccion
```

- **Indentación:** Las instrucciones que pertenecen al `if` deben estar desplazadas hacia la derecha, generalmente con cuatro espacios.
- **Ejemplo:**

```
edad = int(input("¿Cuántos años tienes? "))

if edad >= 13:
```

```
    print("Puedes participar en la actividad.")
```

- **Relación con la vida cotidiana:** “Si está lloviendo, llevo paraguas”.
- **Pruebas:** Ejecutar el programa con valores que hagan verdadera y falsa la condición.

4. Estructuras `if`, `elif` y `else`

- **`if`:** Comprueba la primera condición.
- **`elif`:** Comprueba una condición alternativa si la anterior no se cumplió. Puede utilizarse más de una vez.
- **`else`:** Ejecuta instrucciones cuando ninguna de las condiciones anteriores es verdadera.
- **Ejemplo de clasificación de una calificación:**

```
nota = float(input("Escribe tu calificación: "))

if nota >= 9:
    print("Excelente")
elif nota >= 7:
    print("Aprobado")
else:
    print("Necesitas seguir practicando")
```

- **Orden de evaluación:** Python revisa las condiciones de arriba hacia abajo y ejecuta únicamente el primer bloque que resulte verdadero.
- **Condicionales anidados:** Un `if` puede estar dentro de otro, aunque se debe evitar una complejidad innecesaria.
- **Buenas prácticas:** Usar mensajes claros, condiciones ordenadas y nombres de variables comprensibles.

5. Ciclo `for`: repetir una cantidad conocida de veces

- **Descripción:** El ciclo `for` repite un bloque de instrucciones recorriendo una secuencia o un rango de valores.
- **Función `range()`:** Genera una secuencia de números para controlar las repeticiones.
- **Ejemplo:**

```
for numero in range(1, 6):
    print(numero)
```

- **Repetición de mensajes:**

```
for repeticion in range(3):
    print("¡Python es creativo!")
```

- **Recorrido de una lista:**

```
colores = ["azul", "verde", "rojo"]

for color in colores:
    print(color)
```

- **Variables de control:** La variable del ciclo toma cada valor de la secuencia.
- **Ventaja:** Evita escribir muchas veces la misma instrucción.

- **Cálculos acumulados:**

```
suma = 0

for numero in range(1, 6):
    suma = suma + numero

print("La suma es:", suma)
```

6. Ciclo `while`: repetir mientras se cumpla una condición

- **Descripción:** El ciclo `while` repite instrucciones mientras una condición sea verdadera.
- **Sintaxis básica:**

```
while condicion:
    instrucciones
```

- **Ejemplo de contador:**

```
contador = 1

while contador = 5:
    print(contador)
    contador = contador + 1
```

- **Actualización de la condición:** Es necesario modificar alguna variable dentro del ciclo para evitar un ciclo infinito.
- **Menú interactivo:**

```
opcion = ""

while opcion != "3":
    print("1. Saludar")
    print("2. Mostrar un mensaje")
    print("3. Salir")
    opcion = input("Elige una opción: ")

    if opcion == "1":
        print("¡Hola!")
    elif opcion == "2":
        print("Estás aprendiendo Python.")
    elif opcion != "3":
        print("Opción no válida")
```

- **Uso apropiado:** Cuando no se conoce exactamente cuántas veces se repetirá una acción, por ejemplo, hasta que el usuario escriba una opción de salida.

7. Comparación entre `for` y `while`

- **`for`:** Se recomienda cuando se conoce la cantidad de repeticiones o se desea recorrer una lista, texto o rango.
- **`while`:** Se recomienda cuando la repetición depende de una condición que puede cambiar.
- **Pregunta orientadora:** “¿Sé cuántas veces debo repetir?” Si la respuesta es sí, probablemente conviene usar `for`.

- **Errores comunes:** No actualizar la variable de control, colocar mal la indentación o utilizar una condición que nunca puede cumplirse.
- **Detención anticipada:** Introducción opcional a `break` para salir de un ciclo cuando se cumple una condición.

8. Combinación de decisiones y ciclos

- **Descripción:** Se integrarán condicionales y ciclos para resolver problemas interactivos.
- **Ejemplo de conteo:**

```
positivos = 0

for i in range(5):
    numero = int(input("Escribe un número: "))

    if numero > 0:
        positivos = positivos + 1

print("Escribiste", positivos, "números positivos.")
```

- **Ejemplo de juego sencillo:** El programa puede pedir una respuesta varias veces y dar pistas mediante `if`, `elif` y `else`.
- **Validación de datos:** Comprobar que una opción ingresada pertenezca a las alternativas disponibles.
- **Diseño por etapas:** Definir el problema, identificar entradas y salidas, escribir un algoritmo, programar, probar y corregir.
- **Pruebas:** Utilizar al menos tres casos: un caso normal, un caso límite y un caso que genere una respuesta diferente.

9. Pruebas, depuración y explicación del programa

- **Prueba:** Ejecutar el programa con datos seleccionados para comprobar si produce el resultado esperado.
- **Tabla de pruebas:** Registrar entrada, resultado esperado, resultado obtenido y observaciones.
- **Errores de sintaxis:** Ocurren cuando se escribe incorrectamente una instrucción, se olvida un signo o se utiliza mal la indentación.
- **Errores lógicos:** El programa funciona, pero entrega un resultado incorrecto porque la condición o el cálculo no fueron diseñados adecuadamente.
- **Depuración:** Leer el mensaje de error, ubicar la línea indicada, revisar variables y probar nuevamente.
- **Explicación oral o escrita:** El estudiante debe describir qué datos recibe el programa, qué decisiones toma, qué ciclos utiliza y cómo corrigió al menos un error.

Actividades

Actividad 1. “Decisiones de la vida cotidiana”

Objetivo: Contribuye al objetivo de explicar el funcionamiento de `if`, `elif` y `else` mediante situaciones cotidianas.

Descripción paso a paso:

- El docente presenta situaciones como: “Si la temperatura es menor que 15 grados, uso abrigo; si está entre 15 y 25, uso una chaqueta; de lo contrario, uso ropa ligera”.
- En parejas, los estudiantes identifican las condiciones, las posibles respuestas y el caso que corresponde a `else`.
- Cada pareja convierte una situación en un diagrama sencillo o pseudocódigo.
- Después, escriben un programa de Python que solicite un dato al usuario y muestre una recomendación.
- Prueban el programa con al menos tres valores diferentes y explican por qué aparece cada respuesta.

Organización: Parejas.

Producto esperado: Diagrama o pseudocódigo, programa funcional y tabla con tres casos de prueba.

Duración estimada: 60 minutos.

Actividad 2. “Clasificador de resultados”

Objetivo: Contribuye al objetivo de aplicar estructuras condicionales a datos ingresados por el usuario y obtener resultados correctos en al menos tres casos de prueba.

Descripción paso a paso:

- El docente plantea el reto: crear un programa que reciba una calificación de 0 a 10 y la clasifique como “Excelente”, “Aprobado” o “Necesita mejorar”.
- Los estudiantes identifican los rangos y los escriben antes de programar.
- Elaboran el código utilizando `if`, `elif` y `else`.
- Incluyen una condición adicional para detectar valores fuera del rango permitido.
- Prueban el programa con una calificación alta, una intermedia, una baja y un valor inválido.
- Intercambian el programa con otro compañero, quien revisa si las condiciones están ordenadas correctamente.

Organización: Individual, con revisión entre pares.

Producto esperado: Programa clasificador, tabla de pruebas y breve explicación de cada condición.

Duración estimada: 75 minutos.

Actividad 3. “For o while: el ciclo adecuado”

Objetivo: Contribuye al objetivo de utilizar ciclos `for` y `while` para repetir instrucciones evitando código innecesario.

Descripción paso a paso:

- El docente muestra dos problemas: imprimir los números del 1 al 10 y pedir una contraseña hasta que sea correcta.
- Los estudiantes deciden qué ciclo resulta más conveniente en cada caso y justifican su elección.
- Programan el primer problema con `for` utilizando `range()`.
- Programan el segundo problema con `while`, asegurándose de actualizar la variable o condición necesaria.
- Comparan los programas con una solución que repita instrucciones manualmente y explican por qué el ciclo es más eficiente.
- Modifican uno de los programas para cambiar la cantidad de repeticiones o el mensaje de salida.

Organización: Parejas.

Producto esperado: Dos programas funcionales y una explicación escrita de la elección de cada ciclo.

Duración estimada: 75 minutos.

Actividad 4. “Proyecto: menú interactivo para mi contexto”

Objetivo: Contribuye al objetivo de construir y probar un programa interactivo que combine condicionales y ciclos, explique su funcionamiento y corrija al menos un error.

Descripción paso a paso:

- En equipos, los estudiantes seleccionan un problema cercano, por ejemplo: menú de hábitos saludables, recomendador de actividades, registro de puntos de un juego, quiz de conocimientos o control sencillo de gastos.
- Definen el propósito del programa, los datos de entrada, las opciones del usuario y los resultados esperados.
- Diseñan un algoritmo o diagrama de flujo que incluya al menos una decisión y un ciclo.
- Programan un menú interactivo con opciones, mensajes claros y una opción para salir.
- Utilizan `if`, `elif` y `else`, además de un ciclo `for` o `while`.
- Realizan al menos tres casos de prueba y registran los resultados.
- Provocan o identifican un error de sintaxis o de lógica, explican su causa y documentan la corrección.
- Presentan el programa al grupo y explican qué hace cada estructura de control.

Organización: Equipos de tres o cuatro estudiantes.

Producto esperado: Programa interactivo, algoritmo o diagrama, tabla de pruebas, registro de un error corregido y presentación breve.

Duración estimada: 150 minutos.

Evaluación

Evaluación diagnóstica

Propósito: Identificar los conocimientos previos sobre variables, entrada de datos, comparaciones, secuencias y repetición.

Qué se evalúa:

- Reconocimiento de situaciones en las que se debe tomar una decisión.
- Diferencia entre una instrucción que se ejecuta una vez y una que debe repetirse.
- Comprensión inicial de variables y valores numéricos o de texto.
- Lectura básica de un fragmento corto de código.

Cómo se evalúa: Aplicar al inicio un cuestionario breve de cinco a ocho preguntas y una actividad de predicción. Por ejemplo, pedir a los estudiantes que indiquen qué mensaje mostraría un programa con un `if`.

Instrumento sugerido: Cuestionario diagnóstico y lista de cotejo de conocimientos previos. Esta evaluación no debe calificarse como nota definitiva; sirve para ajustar el ritmo y formar parejas o equipos.

Evaluación formativa

Propósito: Acompañar el aprendizaje durante la práctica y detectar errores antes del proyecto final.

Qué se evalúa:

- Uso correcto de operadores de comparación y operadores lógicos.
- Aplicación de `if`, `elif` y `else` en el orden adecuado.
- Uso correcto de la indentación.
- Elección justificada entre `for` y `while`.
- Actualización de la variable de control en un ciclo `while`.
- Capacidad para predecir resultados y localizar errores.

Cómo se evalúa:

- Revisión de los programas realizados en las actividades.
- Preguntas orales durante la programación: “¿Qué ocurre si el usuario escribe este valor?” o “¿Cuándo termina el ciclo?”.
- Ejercicios de completar código y corregir errores.
- Revisión entre pares de tablas de pruebas y condiciones.
- Retroalimentación breve del docente utilizando ejemplos concretos.

Instrumentos sugeridos: Lista de cotejo, guía de observación, ejercicios de depuración y ficha de autoevaluación.

Evaluación sumativa

Propósito: Valorar el logro integrado de los objetivos de la unidad mediante un programa funcional y una explicación de su funcionamiento.

Producto evaluado: Proyecto “Menú interactivo para mi contexto”.

Criterios de evaluación:

- **Condicionales:** Utiliza correctamente `if`, `elif` y `else` para tomar decisiones.
- **Entrada y salida:** Solicita datos de manera clara y muestra resultados comprensibles.
- **Ciclos:** Usa adecuadamente `for` o `while` para evitar repeticiones innecesarias.
- **Interactividad:** El usuario puede elegir opciones, recibir respuestas y salir del programa.
- **Pruebas:** Presenta al menos tres casos de prueba con resultados correctos.
- **Depuración:** Identifica y corrige al menos un error, explicando qué lo provocaba.
- **Explicación:** Describe el propósito del programa y el funcionamiento de sus decisiones y ciclos.
- **Trabajo colaborativo:** Participa responsablemente y comunica sus aportes al equipo.

Instrumento sugerido: Rúbrica analítica de 4 niveles:

- **4 - Sobresaliente:** El programa funciona correctamente, combina las estructuras solicitadas, supera las pruebas y la explicación es clara y completa.
- **3 - Logrado:** El programa funciona con pequeños errores, utiliza las estructuras principales y presenta pruebas suficientes.
- **2 - En proceso:** El programa funciona parcialmente o presenta errores importantes en condiciones, ciclos o pruebas.
- **1 - Inicial:** El programa está incompleto y requiere apoyo constante para aplicar decisiones, ciclos y pruebas.

Evidencias complementarias: Código fuente, diagrama o pseudocódigo, tabla de casos de prueba, registro de depuración y exposición de tres a cinco minutos.

Unidad 4: Funciones y proyecto integrador

Objetivos de Aprendizaje

- Al finalizar la unidad, el estudiante será capaz de definir y utilizar funciones en Python para organizar y reutilizar código, incorporando parámetros y valores de retorno en ejercicios sencillos.
- Al finalizar la unidad, el estudiante será capaz de descomponer un problema cotidiano en tareas más pequeñas y convertirlas en funciones, siguiendo una secuencia lógica y utilizando nombres descriptivos.
- Al finalizar la unidad, el estudiante será capaz de diseñar y programar un proyecto sencillo en Python que responda a una necesidad o problema de su contexto, integrando variables, estructuras condicionales, ciclos, funciones y entrada y salida de datos.
- Al finalizar la unidad, el estudiante será capaz de probar, explicar y mejorar su proyecto mediante casos de prueba, identificación y corrección de errores, y una presentación clara de su funcionamiento y propósito.

Contenidos Temáticos

1. Las funciones como herramientas para organizar programas

Se introducirá el concepto de función como un bloque de instrucciones que realiza una tarea específica. Se relacionará con acciones conocidas, como preparar una receta, calcular un promedio o mostrar un menú.

- **¿Qué es una función?** Un conjunto de instrucciones con un nombre que puede ejecutarse cuando se necesite.
- **Ventajas de utilizar funciones:** organización, reutilización, legibilidad, facilidad para probar y corregir errores.
- **Partes de una función:** palabra clave `def`, nombre, paréntesis, parámetros opcionales, dos puntos e instrucciones indentadas.
- **Definición y llamada de funciones.**
- **Uso de nombres descriptivos:** `mostrar_menu()`, `calcular_promedio()`, `convertir_temperatura()`.
- **Indentación:** importancia de mantener las instrucciones de la función con el mismo nivel de sangría.

```
def saludar():
    print("¡Hola! Bienvenido al programa.")
```

```
saludar()
```

2. Parámetros y argumentos

Se aprenderá a crear funciones que reciban información para realizar tareas más flexibles y reutilizables.

- **Parámetros:** variables que se escriben en la definición de la función.
- **Argumentos:** valores que se envían cuando se llama a la función.
- **Funciones con un parámetro.**
- **Funciones con varios parámetros.**
- **Orden de los argumentos:** relación entre la posición de los argumentos y los parámetros.
- **Tipos de datos en los parámetros:** textos, números y valores booleanos.
- **Errores frecuentes:** olvidar un argumento, escribir mal el nombre o cambiar el orden de los valores.

```
def saludar_persona(nombre):
    print("Hola,", nombre)

saludar_persona("Camila")
saludar_persona("Diego")
```

3. Valores de retorno

Se explicará la diferencia entre mostrar un resultado en pantalla y devolverlo para utilizarlo posteriormente en otra parte del programa.

- **Instrucción `return`:** devuelve un resultado desde una función.
- **Diferencia entre `print()` y `return`:** `print()` muestra información; `return` entrega un valor al programa.
- **Almacenamiento del resultado:** guardar el valor retornado en una variable.
- **Uso de valores retornados en cálculos y condiciones.**
- **Funciones que retornan números, textos o valores booleanos.**
- **Buenas prácticas:** una función debe realizar una tarea clara y, cuando sea posible, tener una responsabilidad principal.

```
def sumar(numero1, numero2):
    resultado = numero1 + numero2
    return resultado

total = sumar(8, 5)
print("El total es:", total)
```

4. Variables locales y alcance básico

Se abordará de manera introductoria dónde se pueden utilizar las variables creadas dentro de una función y cómo enviar información entre funciones.

- **Variables locales:** variables creadas dentro de una función.
- **Parámetros como datos de entrada de una función.**

- **Valores de retorno como datos de salida.**
- **Evitar depender innecesariamente de variables globales.**
- **Flujo de información:** entrada, procesamiento y salida.

5. Descomposición de problemas cotidianos

Se enseñará a analizar una necesidad o problema, dividirlo en tareas pequeñas y convertir cada tarea en una función.

- **Identificación del problema:** ¿qué se necesita resolver?, ¿para quién?, ¿qué resultado se espera?
- **Identificación de entradas:** datos que el usuario debe proporcionar.
- **Identificación de procesos:** cálculos, decisiones o repeticiones necesarias.
- **Identificación de salidas:** información que el programa mostrará o entregará.
- **División en tareas:** separar el problema en acciones pequeñas y ordenadas.
- **Diseño de funciones:** asignar una función a cada tarea importante.
- **Secuencia lógica:** determinar qué función debe ejecutarse primero, después y al final.
- **Uso de diagramas de flujo, listas de pasos o pseudocódigo.**

Ejemplo de descomposición para una aplicación que calcula el costo de una merienda:

- `mostrar_menu()`: presenta las opciones.
- `leer_opcion()`: solicita una opción al usuario.
- `calcular_total(precio, cantidad)`: calcula el costo.
- `aplicar_descuento(total)`: determina si corresponde un descuento.
- `mostrar_resumen(total, descuento)`: presenta el resultado.

6. Integración de elementos fundamentales de Python

Se integrarán los contenidos estudiados previamente en programas más completos.

- **Variables:** almacenamiento de nombres, cantidades, puntajes y resultados.
- **Entrada de datos:** uso de `input()`.
- **Conversión de tipos:** uso de `int()` y `float()` cuando sea necesario.
- **Salida de datos:** uso de `print()` y mensajes claros.
- **Estructuras condicionales:** `if`, `elif` y `else`.
- **Ciclos:** uso de `while` para repetir menús o validar datos y `for` para recorrer elementos.
- **Funciones:** organización del programa en tareas reutilizables.
- **Validación básica:** comprobación de opciones válidas y datos positivos.

7. Diseño del proyecto integrador

Los estudiantes diseñarán un programa sencillo que responda a una necesidad o problema de su contexto escolar, familiar o comunitario.

- **Selección del problema:** debe ser concreto, comprensible y posible de resolver con los conocimientos de la unidad.
- **Ejemplos de proyectos:** calculadora de ahorro, organizador de tareas, cuestionario educativo, registro de gastos, calculadora de promedio, menú de recetas, sistema de turnos sencillo o juego de preguntas.
- **Propósito del programa:** explicación breve de qué problema resuelve y a quién beneficia.
- **Usuarios y contexto:** personas que utilizarían el programa y situación en la que sería útil.
- **Entradas, procesos y salidas.**
- **Lista de funciones:** nombre, propósito, parámetros y valor de retorno, si corresponde.
- **Planificación mediante pseudocódigo o diagrama de flujo.**
- **Criterios mínimos del proyecto:** incluir variables, entrada y salida de datos, al menos una condición, al menos un ciclo y al menos tres funciones.

8. Programación y documentación del proyecto

Se construirá el proyecto paso a paso, probando cada función antes de integrarla al programa completo.

- Crear primero las funciones principales.
- Probar cada función con datos sencillos.
- Crear una función principal o un bloque de inicio que coordine el programa.
- Utilizar nombres claros para funciones y variables.
- Agregar comentarios únicamente cuando ayuden a comprender una parte importante del código.
- Mostrar mensajes que indiquen al usuario qué debe hacer.
- Evitar repetir código cuando pueda reutilizarse una función.
- Organizar el código con indentación y separación visual.

9. Pruebas, depuración y mejora

Se aprenderá a verificar el funcionamiento del programa, encontrar errores y realizar mejoras a partir de evidencias.

- **Pruebas normales:** datos esperados y correctos.
- **Pruebas límite:** valores mínimos, máximos o cercanos a ellos.
- **Pruebas de error:** opciones inválidas, textos inesperados o valores negativos.
- **Tabla de casos de prueba:** entrada, resultado esperado, resultado obtenido y observaciones.
- **Errores de sintaxis:** problemas con signos, indentación o nombres.
- **Errores de ejecución:** problemas que aparecen mientras el programa funciona.
- **Errores lógicos:** el programa funciona, pero entrega un resultado incorrecto.
- **Depuración:** leer el mensaje de error, localizar la línea, formular una posible causa, modificar y volver a probar.
- **Mejora del programa:** mensajes más claros, validación de datos, mejor organización o nuevas opciones.

10. Comunicación y presentación del proyecto

Los estudiantes explicarán el propósito, funcionamiento y proceso de creación de su programa.

- Presentar el problema o necesidad que motivó el proyecto.
- Explicar quiénes son los posibles usuarios.
- Mostrar las funciones principales.
- Realizar una demostración con un caso de uso.
- Explicar al menos un error encontrado y cómo fue corregido.
- Comentar una mejora realizada o una mejora futura.
- Responder preguntas de los compañeros y del docente.

Actividades

Actividad 1: “Mis primeras funciones reutilizables”

Objetivo: Contribuye al objetivo de definir y utilizar funciones en Python, incorporando parámetros y valores de retorno en ejercicios sencillos.

Descripción paso a paso:

- El docente presenta una situación cotidiana, como preparar una bebida, saludar a diferentes personas o calcular el costo de varios productos.
- Los estudiantes identifican qué instrucciones se repiten y cuáles pueden agruparse.
- El docente demuestra la diferencia entre definir una función y llamarla.
- Cada estudiante crea una función sin parámetros, por ejemplo `mostrar_instrucciones()`.
- Después crea una función con un parámetro, por ejemplo `saludar(nombre)`.
- Finalmente crea una función con parámetros y retorno, por ejemplo `calcular_area(base, altura)`.
- Los estudiantes prueban cada función con al menos tres valores diferentes.
- Se realiza una revisión por parejas utilizando una lista de comprobación: nombre claro, indentación correcta, llamada funcional y resultado esperado.

Organización: Trabajo individual y revisión en parejas.

Producto esperado: Archivo de Python con tres funciones: una sin parámetros, una con parámetros y una con valor de retorno, acompañadas de pruebas.

Duración estimada: 90 minutos.

Actividad 2: “Del problema a las funciones”

Objetivo: Contribuye al objetivo de descomponer un problema cotidiano en tareas pequeñas y convertirlas en funciones con nombres descriptivos.

Descripción paso a paso:

- El docente presenta el problema: “Una familia quiere organizar sus gastos semanales y saber cuánto dinero le queda”.

- En grupos pequeños, los estudiantes responden: ¿qué datos se necesitan?, ¿qué cálculos deben realizarse?, ¿qué información debe mostrarse?
- Cada grupo escribe el problema como una secuencia de pasos.
- Los estudiantes separan los pasos en tareas: solicitar ingresos, registrar gastos, calcular total, calcular saldo y mostrar un resumen.
- Para cada tarea proponen un nombre de función, por ejemplo `pedir_ingreso()`, `calcular_total_gastos()` y `mostrar_resumen()`.
- El grupo indica qué parámetros necesita cada función y qué funciones deberían devolver un resultado.
- Representan la solución mediante un diagrama de flujo o pseudocódigo.
- Cada grupo explica su propuesta y recibe sugerencias de los demás.

Organización: Grupos de tres o cuatro estudiantes.

Producto esperado: Ficha de diseño con problema, entradas, procesos, salidas, lista de funciones, parámetros, retornos y secuencia lógica.

Duración estimada: 90 minutos.

Actividad 3: “Laboratorio de integración”

Objetivo: Contribuye al objetivo de integrar variables, condiciones, ciclos, funciones, entrada y salida de datos en un programa sencillo.

Descripción paso a paso:

- El docente entrega o proyecta el reto: crear un menú de opciones para una calculadora de promedio.
- El programa debe permitir ingresar varias calificaciones, calcular el promedio y determinar si el estudiante aprobó.
- Los estudiantes identifican las funciones necesarias, por ejemplo:
  - `mostrar_menu()`
  - `ingresar_calificaciones()`
  - `calcular_promedio(calificaciones)`
  - `determinar_resultado(promedio)`
- Programan el menú utilizando un ciclo `while`.
- Utilizan una condición para determinar el resultado según el promedio.
- Prueban el programa con calificaciones aprobatorias, no aprobatorias y valores límite.
- Comparan su código con una lista de criterios y realizan una mejora.

Organización: Parejas.

Producto esperado: Programa funcional de consola que utilice al menos tres funciones, un ciclo, una condición, entradas y salidas.

Duración estimada: 120 minutos.

Actividad 4: “Feria de ideas para el proyecto integrador”

Objetivo: Contribuye al objetivo de diseñar un proyecto que responda a una necesidad o problema del contexto del estudiante.

Descripción paso a paso:

- Cada estudiante o pareja realiza una lluvia de ideas sobre problemas de la escuela, hogar o comunidad que podrían resolverse con un programa sencillo.
- Clasifican las ideas según su utilidad, dificultad y posibilidad de realizarlas con los contenidos estudiados.
- Seleccionan una propuesta y completan una ficha de proyecto.
- La ficha debe incluir nombre del proyecto, problema, usuarios, propósito, entradas, procesos, salidas y funciones previstas.
- El docente revisa que el proyecto sea adecuado para el tiempo disponible y que incluya los elementos mínimos.
- Cada pareja presenta su idea en un minuto.
- Los compañeros ofrecen una sugerencia y una pregunta para ayudar a mejorar la propuesta.

Organización: Individual o en parejas.

Producto esperado: Propuesta aprobada del proyecto integrador y esquema inicial de funciones.

Duración estimada: 90 minutos.

Actividad 5: “Construcción y pruebas del proyecto”

Objetivo: Contribuye al objetivo de programar, probar y mejorar un proyecto integrando los elementos fundamentales de Python.

Descripción paso a paso:

- Los estudiantes transforman su ficha de diseño en pseudocódigo o diagrama de flujo.
- Crean las funciones principales de manera independiente.
- Prueban cada función con datos sencillos antes de unirlas.
- Integran las funciones en un programa completo.
- Incorporan un menú o una secuencia de interacción con el usuario.
- Agregan condiciones para tomar decisiones y ciclos para repetir acciones cuando sea necesario.
- Elaboran una tabla con al menos cinco casos de prueba: dos normales, uno límite y dos casos de error o entradas inesperadas.
- Registran los errores encontrados, la posible causa, la solución aplicada y el resultado de la nueva prueba.
- Realizan una revisión final de nombres, indentación, mensajes y organización del código.

Organización: Individual o en parejas.

Producto esperado: Programa funcional, tabla de casos de prueba, registro de depuración y breve documentación del código.

Duración estimada: 180 minutos.

Actividad 6: “Demo y presentación del proyecto”

Objetivo: Contribuye al objetivo de explicar el funcionamiento del proyecto, comunicar su propósito y proponer mejoras.

Descripción paso a paso:

- Cada estudiante o pareja prepara una presentación de tres a cinco minutos.
- La presentación debe explicar el problema, los usuarios, el funcionamiento, las funciones principales y los resultados de las pruebas.
- Los estudiantes realizan una demostración del programa utilizando un caso de uso preparado.
- Explican un error que encontraron y la forma en que lo solucionaron.
- Indican una mejora realizada y una posible mejora futura.
- Los compañeros completan una ficha de retroalimentación con dos aspectos positivos y una sugerencia.
- Después de la presentación, cada estudiante escribe una reflexión breve sobre lo que aprendió y lo que cambiaría en una nueva versión.

Organización: Individual o en parejas; presentación ante el grupo.

Producto esperado: Presentación oral, demostración del programa y reflexión individual.

Duración estimada: 120 minutos.

Evaluación

Evaluación diagnóstica

Propósito: Identificar los conocimientos previos sobre variables, entrada y salida de datos, condiciones, ciclos y organización de programas.

Qué se evalúa:

- Comprensión de la secuencia de instrucciones.
- Uso básico de variables, `input()` y `print()`.
- Reconocimiento de condiciones y ciclos.
- Capacidad para explicar qué hace un fragmento corto de código.
- Ideas iniciales sobre la utilidad de una función.

Cómo se evalúa: Aplicar al inicio un cuestionario breve de cinco a ocho preguntas y un reto práctico, como completar un programa que solicite dos números y muestre el mayor.

Instrumento sugerido: Cuestionario diagnóstico y lista de cotejo inicial. Esta evaluación no debe tener una calificación sumativa; se utiliza para ajustar la explicación y organizar apoyos.

Evaluación formativa

Propósito: Acompañar el aprendizaje durante la creación de funciones, la descomposición de problemas y la construcción del proyecto.

Qué se evalúa:

- Define funciones utilizando correctamente `def`, paréntesis, dos puntos e indentación.
- Utiliza nombres descriptivos para funciones y variables.
- Envía parámetros correctamente.
- Utiliza `return` cuando necesita entregar un resultado.
- Explica la diferencia entre mostrar y retornar un valor.
- Divide un problema en tareas claras y ordenadas.
- Relaciona entradas, procesos y salidas.
- Prueba las funciones con diferentes datos.
- Participa en la revisión por pares y utiliza la retroalimentación.
- Identifica errores de sintaxis, ejecución o lógica y propone soluciones.

Cómo se evalúa: Mediante observación durante las actividades, revisión de ejercicios, preguntas orales, análisis de pseudocódigos, comprobación de avances y conversaciones breves con cada estudiante o pareja.

Instrumentos sugeridos:

- Lista de cotejo para funciones.
- Guía de revisión por parejas.
- Diario o registro de errores.
- Ticket de salida con preguntas como: “¿Cuándo usarías `return`?” y “¿Qué tarea convertirías en una función?”.
- Tabla de seguimiento del proyecto.

Evaluación sumativa

Propósito: Valorar el dominio integrado de los objetivos mediante un proyecto funcional y su presentación.

Qué se evalúa:

- **Diseño del proyecto:** el problema es claro, pertinente y posible de resolver.
- **Descomposición:** el programa está dividido en tareas y funciones con una secuencia lógica.
- **Uso de funciones:** incluye al menos tres funciones, con nombres descriptivos y parámetros o valores de retorno cuando son necesarios.
- **Integración técnica:** utiliza variables, entrada y salida de datos, al menos una condición y al menos un ciclo.
- **Funcionamiento:** el programa cumple su propósito principal y responde adecuadamente a las entradas.
- **Pruebas y depuración:** presenta casos de prueba variados, identifica errores y documenta correcciones.
- **Claridad del código:** utiliza indentación, organización y mensajes comprensibles.
- **Comunicación:** explica el propósito, funcionamiento, funciones principales, pruebas y mejoras.

Cómo se evalúa: El docente revisa el código, observa la demostración, analiza la tabla de pruebas y escucha la presentación. También puede incorporar la autoevaluación y la coevaluación de los compañeros.

Instrumento sugerido: Rúbrica de proyecto con escala de 1 a 4.

- **4 - Sobresaliente:** cumple completamente el criterio, funciona de manera consistente y puede explicarlo con claridad.
- **3 - Logrado:** cumple el criterio con pequeños detalles por mejorar.
- **2 - En proceso:** cumple parcialmente el criterio y necesita apoyo o correcciones.
- **1 - Inicial:** presenta dificultades importantes o evidencia insuficiente.

Propuesta de ponderación:

- Diseño y descomposición del problema: 15%.
- Uso de funciones, parámetros y retornos: 25%.
- Integración de variables, condiciones, ciclos y entrada/salida: 20%.
- Pruebas, depuración y mejoras: 20%.
- Presentación y explicación: 15%.
- Reflexión individual y trabajo responsable: 5%.

Instrumentos complementarios para la evaluación final

- **Lista de cotejo del código:** verifica funciones, parámetros, retornos, variables, condiciones, ciclos y entradas/salidas.
- **Tabla de casos de prueba:** registra entradas, resultados esperados, resultados obtenidos y correcciones.
- **Guía de autoevaluación:** permite responder qué aprendió, qué error corrigió y qué mejoraría.
- **Ficha de coevaluación:** valora la claridad de la presentación, la utilidad del proyecto y la explicación del código.