Curso de Python 2023

Lcdo. Diego Saavedra Mgtr.

2023-08-28

Table of contents

1 Prefacio

¡Bienvenidos al Curso Completo de Python: Desde Fundamentos hasta Aplicaciones Prácticas!

1.1 ¿Qué es este Curso?

Este curso exhaustivo te llevará desde los fundamentos básicos de la programación hasta la creación de aplicaciones prácticas utilizando el lenguaje de programación Python. A través de una combinación de teoría y ejercicios prácticos, te sumergirás en los conceptos esenciales de la programación y avanzarás hacia la construcción de proyectos reales. Desde la instalación de herramientas hasta la creación de una API con Django Rest Framework, este curso te proporcionará una comprensión sólida y práctica de Python y su aplicación en el mundo real.

1.2 ¿A quién está dirigido?

Este curso está diseñado para principiantes y aquellos con poca o ninguna experiencia en programación. No importa si eres un estudiante curioso, un profesional que busca cambiar de carrera o simplemente alguien que desea aprender a programar: este curso es para ti. Desde adolescentes hasta adultos, todos son bienvenidos a participar y explorar el emocionante mundo de la programación a través de Python.

1.3 ¿Cómo contribuir?

Valoramos tu participación en este curso. Si encuentras errores, deseas sugerir mejoras o agregar contenido adicional, ¡nos encantaría escucharte! Puedes contribuir a través de nuestra plataforma en línea, donde puedes compartir tus comentarios y sugerencias. Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de estudiantes y entusiastas de la programación.

Este libro ha sido creado con el objetivo de brindar acceso gratuito y universal al conocimiento. Estará disponible en línea para que cualquiera, sin importar su ubicación o circunstancias, pueda acceder y aprender a su propio ritmo.

¡Esperamos que disfrutes este emocionante viaje de aprendizaje y descubrimiento en el mundo de la programación con Python!

2 Introducción a la programación.

2.1 Contenido

La programación es el proceso de crear secuencias de instrucciones que le indican a una computadora cómo realizar una tarea específica.

Estas instrucciones se escriben en lenguajes de programación, que son conjuntos de reglas y símbolos utilizados para comunicarse con la máquina. La programación es una habilidad esencial en la era digital, ya que se aplica en una amplia variedad de campos, desde desarrollo de software y análisis de datos hasta diseño de juegos y automatización.

2.2 Conceptos Clave

2.2.1 Instrucciones

Son comandos específicos que le indican a la computadora qué hacer. Pueden ser simples, como imprimir un mensaje en pantalla, o complejas, como realizar cálculos matemáticos.

2.2.2 Lenguajes de Programación

Son sistemas de comunicación entre humanos y máquinas. Cada lenguaje tiene reglas sintácticas y semánticas que determinan cómo se escriben y ejecutan las instrucciones.

2.2.3 Algoritmos

Son conjuntos ordenados de instrucciones diseñados para resolver un problema específico. Los algoritmos son la base de la programación y se utilizan para desarrollar software eficiente.

2.2.4 Depuración

Es el proceso de identificar y corregir errores en el código. Los programadores pasan tiempo depurando para asegurarse de que sus programas funcionen correctamente.

Ejemplo:

```
print("Hola, bienvenido al mundo de la programación.")                    ①
```

① Este es un ejemplo sencillo de un programa en Python que imprime un mensaje en pantalla.

2.2.5 Explicación

En Python, los comentarios comienzan con el símbolo `#`. No afectan la ejecución del programa, pero son útiles para documentar el código.

La línea `print("Hola, bienvenido al mundo de la programación.")` es una instrucción de impresión. La función `print()` muestra el texto entre paréntesis en la consola.

> **!** Actividad Práctica
>
> Escribe un programa que solicite al usuario su nombre y luego imprima un mensaje de bienvenida personalizado.

2.2.6 Explicación de la Actividad

El programa utilizará la función `input()` para recibir la entrada del usuario. Luego, utilizará la entrada proporcionada para imprimir un mensaje de bienvenida personalizado.

3 Instalación de Python y más herramientas

3.1 Contenido:

La instalación de Python es el primer paso para comenzar a programar en este lenguaje. Python es un lenguaje de programación versátil y ampliamente utilizado, conocido por su sintaxis clara y legible. Aquí aprenderemos cómo instalar Python en diferentes sistemas operativos.

3.2 Conceptos Clave

3.2.1 Python

Lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones web, científicas, de automatización y más.

3.2.2 Interprete

Python es un lenguaje interpretado, lo que significa que se ejecuta línea por línea en tiempo real.

3.2.3 IDE

Los entornos de desarrollo integrados (IDE) como Visual Studio Code (VS Code) o PyCharm brindan herramientas para escribir, depurar y ejecutar código de manera más eficiente.

3.3 Ejemplo

No se necesita código para esta lección, ya que se trata de instrucciones para la instalación de Python en diferentes sistemas operativos.

3.4 Explicación

Para instalar Python en sistemas Windows, macOS y Linux, se pueden seguir las instrucciones detalladas proporcionadas en el sitio web oficial de Python www.python.org/downloads/.

La instalación de Python generalmente incluye el intérprete de Python y una serie de herramientas y bibliotecas estándar que hacen que sea fácil comenzar a programar.

! Actividad Práctica

Instala Python en tu sistema operativo siguiendo las instrucciones del sitio web oficial de Python. Luego, verifica que Python esté correctamente instalado ejecutando el intérprete y escribiendo el siguiente código:

```
print("Python se ha instalado correctamente.")
```

3.5 Explicación de la Actividad

Esta actividad permite a los participantes aplicar lo aprendido instalando Python en su propio sistema y ejecutando un programa sencillo para confirmar que la instalación fue exitosa.

4 Introducción a Python

4.1 Distintas formas de trabajar con Python

4.2 Contenido

de programación versátil que ofrece diferentes formas de interactuar con él. Aprenderemos las dos formas principales de trabajar con Python: **el intérprete interactivo** y los **scripts de Python**.

4.3 Conceptos Clave:

4.3.1 Intérprete Interactivo:

Permite ejecutar instrucciones de Python en tiempo real y ver los resultados inmediatamente en la consola.

4.3.2 Scripts de Python:

Son archivos que contienen una serie de instrucciones de Python que se pueden ejecutar en conjunto.

4.3.3 Ambientes Virtuales

Son entornos aislados que permiten tener instalaciones y bibliotecas de Python separadas para diferentes proyectos.

4.3.4 Ejemplo

```
>>> 2 + 3                                                    ①
5


                                                             ②
numero1 = 5
numero2 = 7
resultado = numero1 + numero2
print("El resultado de la suma es:", resultado)
                                                             ③
```

① Uso del intérprete interactivo
② Ejecución de un script de Python
③ Guarda este código en un archivo llamado “suma.py”

4.4 Explicación

El intérprete interactivo permite ejecutar expresiones de Python directamente en la consola y ver los resultados en tiempo real.

Los scripts de Python son archivos que contienen un conjunto de instrucciones. En este ejemplo, se muestra cómo crear un script simple que calcula la suma de dos números y lo imprime en la consola.

> **!** Actividad Práctica
>
> Abre el intérprete interactivo de Python y realiza algunas operaciones matemáticas simples.
> Crea un archivo llamado “operaciones.py” y escribe un programa que realice operaciones aritméticas básicas y las muestre en la consola.

4.4.1 Explicación de la Actividad

Esta actividad permite a los participantes experimentar con el intérprete interactivo de Python y crear su propio script para realizar operaciones matemáticas.

4.5 Las bases de Python

4.6 Contenido

En esta lección, exploraremos las bases fundamentales de Python. Aprenderemos sobre las variables, tipos de datos y operadores básicos que forman la base de cualquier programa en Python.

4.7 Conceptos Clave

4.7.1 Variables

Son nombres que se utilizan para almacenar valores en la memoria de la computadora.

4.7.2 Tipos de Datos

Incluyen enteros, flotantes, cadenas, booleanos y más. Cada tipo de dato tiene sus propias características y operaciones.

4.7.3 Operadores

Incluyen operadores aritméticos, de comparación y lógicos que se utilizan para realizar diferentes tipos de cálculos y comparaciones.

4.7.4 Ejemplo

```
# Variables y tipos de datos
nombre = "Juan"
edad = 25
altura = 1.75
es_mayor_de_edad = True

# Operadores aritméticos
suma = 5 + 3
resta = 10 - 2
multiplicacion = 4 * 6
division = 15 / 3
```

```
# Operadores de comparación
igual = 5 == 5
mayor_que = 10 > 5
menor_que = 7 < 12

# Operadores lógicos
and_logico = True and False
or_logico = True or False
not_logico = not True
```

4.8 Explicación:

Las variables se utilizan para almacenar información, como el nombre, la edad, la altura y si una persona es mayor de edad.

Los operadores aritméticos se utilizan para realizar cálculos matemáticos básicos.

Los operadores de comparación se utilizan para comparar valores y devuelven un valor booleano (verdadero o falso).

Los operadores lógicos se utilizan para combinar expresiones booleanas y realizar operaciones lógicas.

> ! Actividad Práctica:
>
> Crea variables que almacenen información sobre ti, como tu nombre, edad y altura.
> Realiza operaciones aritméticas y utiliza operadores de comparación para comparar valores.
> Combina expresiones booleanas utilizando operadores lógicos y observa los resultados.

4.9 Explicación de la Actividad

Esta actividad permite a los participantes aplicar los conceptos de variables, tipos de datos y operadores en ejemplos prácticos. Les ayuda a comprender cómo trabajar con diferentes tipos de datos y cómo realizar operaciones básicas en Python.

4.10 Identación

4.11 Contenido

En Python, la identación (sangría) juega un papel crucial en la estructura y organización del código. Aprenderemos cómo se utiliza la identación para delimitar bloques de código y mejorar la legibilidad.

4.12 Conceptos Clave

4.12.1 Identación

- Espacios o tabulaciones al comienzo de una línea que indican la estructura del código.
- Bloques de Código: Conjuntos de instrucciones que se agrupan juntas y se ejecutan en conjunto.
- PEP 8: Guía de estilo para la escritura de código en Python que recomienda el uso de cuatro espacios para la identación.

4.12.2 Ejemplo

```
# Uso de la identación en un condicional
numero = 10

if numero > 5:
    print("El número es mayor que 5")
else:
    print("El número no es mayor que 5")
```

4.13 Explicación

- En este ejemplo, la identación se utiliza para delimitar los bloques de código dentro de las instrucciones “if” y “else”.
- La identación es crucial para que Python sepa qué instrucciones están dentro de un bloque y cuáles están fuera.

❗ Actividad Práctica:

Escribe un programa que solicite al usuario su edad y muestre un mensaje según si es mayor de 18 años o no.
Intenta cambiar la identación incorrectamente y observa cómo afecta al funcionamiento del programa.

4.14 Explicación de la Actividad.

Esta actividad permite a los participantes comprender la importancia de la identación en Python al trabajar con bloques de código como los condicionales. Les ayuda a desarrollar el hábito de utilizar la identación adecuada para mantener el código organizado y legible.

4.15 Comentarios

4.16 Contenido

Los comentarios son una herramienta importante en la programación para añadir explicaciones y notas en el código. Aprenderemos cómo agregar comentarios en Python y cómo pueden mejorar la comprensión del código.

4.17 Conceptos Clave:

4.17.1 Comentarios

Son notas en el código que no se ejecutan y se utilizan para explicar el propósito y funcionamiento de partes del programa.

4.17.2 Comentarios de una línea

Se crean con el símbolo “#” y abarcan una sola línea.

4.17.3 Comentarios de múltiples líneas

Se crean entre triple comillas (“ “ ” o ’ ’ ’) y pueden abarcar múltiples líneas.

4.17.4 Ejemplo

```
# Este es un comentario de una línea

"""
Este es un comentario
de múltiples líneas.
Puede abarcar varias líneas.
"""

numero = 42  # Este comentario está después de una instrucción
```

4.18 Explicación:

Los comentarios son ignorados por el intérprete y no afectan la ejecución del código.

Los comentarios son útiles para documentar el código, explicar partes difíciles de entender o dejar notas para otros programadores.

> **!** Important
>
> Actividad Práctica:
> Escribe un programa que realice una tarea sencilla y agrega comentarios para explicar lo que hace cada parte.
> Escribe un comentario de múltiples líneas que explique el propósito general de tu programa.

4.19 Explicación de la Actividad

Esta actividad permite a los participantes practicar la adición de comentarios en su código para mejorar la legibilidad y la comprensión. Les ayuda a desarrollar la habilidad de documentar su código de manera efectiva para ellos mismos y para otros programadores.

4.20 Variables

4.21 Contenido

Las variables son fundamentales en la programación ya que permiten almacenar y manipular datos. Aprenderemos cómo declarar y utilizar variables en Python.

4.22 Conceptos Clave:

4.22.1 Variables

Nombres que representan ubicaciones de memoria donde se almacenan datos.

4.22.2 Declaración de Variables.

- Asignación de un valor a un nombre utilizando el operador "=".
- Convenciones de Nombres: Siguen reglas para ser descriptivos y seguir una estructura (por ejemplo, letras minúsculas y guiones bajos para espacios).

4.22.3 Ejemplo

```
nombre = "Ana"
edad = 30
saldo_bancario = 1500.75
es_mayor_de_edad = True
```

4.23 Explicación:

En este ejemplo, se declaran variables para almacenar el nombre de una persona, su edad, su saldo bancario y un valor booleano que indica si es mayor de edad.

Los nombres de variables son descriptivos y siguen la convención de nombres recomendada (letras minúsculas y guiones bajos para espacios).

! Actividad Práctica

Crea variables para almacenar información personal, como tu ciudad, tu edad y tu ocupación.
Declara variables para almacenar cantidades numéricas, como el precio de un producto y la cantidad de unidades disponibles.

4.24 Explicación de la Actividad

Esta actividad permite a los participantes practicar la declaración de variables en Python y aplicar el concepto de convenciones de nombres.

Les ayuda a comprender cómo almacenar y acceder a datos utilizando variables descriptivas y significativas.

4.25 Múltiples Variables

4.26 Contenido:

En Python, es posible asignar valores a múltiples variables en una sola línea. Aprenderemos cómo declarar y utilizar múltiples variables de manera eficiente.

4.27 Conceptos Clave:

4.27.1 Asignación Múltiple

Permite asignar valores a varias variables en una línea.

4.27.2 Desempaquetado de Valores

Se pueden asignar valores de una lista o tupla a múltiples variables en una sola operación.

4.27.3 Intercambio de Valores

Se pueden intercambiar los valores de dos variables utilizando asignación múltiple.

4.27.4 Ejemplo

```
nombre, edad, altura = "María", 28, 1.65
productos = ("Manzanas", "Peras", "Uvas")
producto1, producto2, producto3 = productos
```

4.28 Explicación:

En el primer ejemplo, se utilizó la asignación múltiple para declarar tres variables en una sola línea.

En el segundo ejemplo, se desempaquetaron los valores de una tupla en variables individuales.

::: {.callout-important} ### Actividad Práctica:

Crea una lista con los nombres de tus tres colores favoritos.

Utiliza la asignación múltiple para asignar los valores de la lista a tres variables individuales.

4.29 Explicación de la Actividad

Esta actividad permite a los participantes practicar la asignación múltiple y el desempaquetado de valores.

Les ayuda a comprender cómo trabajar eficientemente con múltiples variables y cómo aprovechar estas técnicas para simplificar el código.

4.30 Concatenación

4.31 Contenido:

La concatenación es la unión de cadenas de texto. Aprenderemos cómo combinar cadenas de texto en Python para crear mensajes más complejos.

4.32 Conceptos Clave:

4.32.1 Concatenación:

Operación que combina dos o más cadenas de texto para formar una cadena más larga.

4.32.2 Operador +

Se utiliza para concatenar cadenas de texto.

4.32.3 Conversión a Cadena

Es necesario convertir valores no string a cadenas antes de concatenarlos.

4.32.4 Ejemplo

```
nombre = "Luisa"
mensaje = "Hola, " + nombre + ". ¿Cómo estás?"
edad = 25
mensaje_edad = "Tienes " + str(edad) + " años."
```

4.33 Explicación:

En este ejemplo, se utilizó el operador "+" para concatenar cadenas de texto.

La variable "edad" se convirtió a una cadena utilizando la función "str()" antes de concatenarla.

> ❗ Actividad Práctica:
>
> Crea una variable con tu comida favorita. Utiliza la concatenación para crear un mensaje que incluya tu comida favorita.

4.34 Explicación de la Actividad

Esta actividad permite a los participantes practicar la concatenación de cadenas de texto y comprender cómo construir mensajes más complejos utilizando variables y texto. Les ayuda a mejorar su capacidad para crear mensajes personalizados en sus programas.

5 Tipos de Datos

5.1 String y Números

5.2 Contenido:

Los strings y los números son dos tipos de datos fundamentales en Python. Aprenderemos cómo trabajar con strings (cadenas de texto) y los diferentes tipos de números en Python.

5.3 Conceptos Clave:

5.3.1 String:

Secuencia de caracteres alfanuméricos. Se pueden definir utilizando comillas simples o dobles.

5.3.2 Números Enteros (int)

Representan números enteros positivos o negativos.

5.3.3 Números de Punto Flotante (float)

Representan números con decimales.

5.4 Ejemplo:

```
# Strings
mensaje = "Hola, bienvenido al curso de Python."
nombre = 'María'

# Números
```

```
edad = 25
saldo = 1500.75
```

5.5 Explicación:

En este ejemplo, se crean variables que almacenan strings y números.

Los strings se definen utilizando comillas simples o dobles.

Los números enteros y de punto flotante se asignan directamente a variables.

> ! Actividad Práctica:
>
> Crea una variable con tu canción favorita.
> Asigna tu edad a una variable y tu altura a otra variable.
> Combina las variables para crear un mensaje personalizado.

5.6 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de strings y trabajar con números enteros y de punto flotante.

Les ayuda a comprender cómo almacenar y manipular diferentes tipos de datos en Python.

5.7 Listas

5.7.1 Contenido:

Las listas son estructuras de datos que permiten almacenar varios elementos en una sola variable. Aprenderemos cómo crear y manipular listas en Python.

5.8 Conceptos Clave:

5.8.1 Listas

Secuencias ordenadas de elementos que pueden ser de diferentes tipos.

5.8.2 Índices

Números que indican la posición de un elemento en la lista.

5.8.3 Acceso a Elementos

Se utiliza el índice para acceder a un elemento específico de la lista.

5.9 Ejemplo

```
frutas = ["manzana", "banana", "naranja", "uva"]
primer_fruta = frutas[0]
segunda_fruta = frutas[1]
```

5.10 Explicación:

En este ejemplo, se crea una lista de frutas y se accede a elementos individuales utilizando índices.

Los índices comienzan desde 0, por lo que la primera fruta tiene el índice 0.

> ❗ Actividad Práctica:
>
> Crea una lista con los nombres de tus tres películas favoritas.
> Accede al segundo elemento de la lista e imprímelo en la consola.

5.11 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de listas y el acceso a elementos utilizando índices. Les ayuda a comprender cómo organizar y acceder a múltiples elementos en una sola variable.

5.12 Tuplas

5.12.1 Contenido:

Las tuplas son estructuras de datos similares a las listas, pero son inmutables, lo que significa que no se pueden modificar después de ser creadas. Aprenderemos cómo trabajar con tuplas en Python.

5.13 Conceptos Clave:

5.13.1 Tuplas

Secuencias ordenadas de elementos que, a diferencia de las listas, no se pueden modificar.

5.13.2 Inmutabilidad

Una vez creada una tupla, no se pueden agregar, modificar o eliminar elementos.

5.13.3 Acceso a Elementos

Se utiliza el índice para acceder a un elemento específico de la tupla.

5.14 Ejemplo:

```
coordenadas = (3, 5)
x = coordenadas[0]
y = coordenadas[1]
```

5.15 Explicación:

En este ejemplo, se crea una tupla que almacena coordenadas (x, y) y se accede a los valores individuales utilizando índices.

! Actividad Práctica:

Crea una tupla con las estaciones del año.
Intenta modificar un elemento de la tupla y observa el error que se produce.

5.16 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de tuplas y comprender la diferencia entre listas y tuplas en términos de inmutabilidad. Les ayuda a comprender cómo utilizar tuplas cuando necesitan almacenar datos que no deben cambiar.

5.17 Range

5.17.1 Contenido

El tipo de dato range se utiliza para generar secuencias de números. Aprenderemos cómo utilizar range en Python para crear secuencias de números en rangos específicos.

5.18 Conceptos Clave:

5.18.1 range

Tipo de dato utilizado para generar secuencias de números en un rango.

5.18.2 Parámetros de range

Se pueden especificar el valor inicial, valor final y paso de la secuencia.

5.18.3 Conversión a Listas

Es posible convertir un objeto range en una lista utilizando la función list().

5.19 Ejemplo

```
# Generación de secuencias de números
secuencia1 = range(5)          # 0, 1, 2, 3, 4
secuencia2 = range(2, 10)      # 2, 3, 4, 5, 6, 7, 8, 9
secuencia3 = range(1, 11, 2)   # 1, 3, 5, 7, 9

# Conversión a lista
lista_secuencia1 = list(secuencia1)
```

5.20 Explicación:

En este ejemplo, se utilizan diferentes valores para crear secuencias de números utilizando el tipo de dato range.

La función list() se utiliza para convertir una secuencia de range en una lista.

> ! Actividad Práctica:
>
> Crea una secuencia de números del 10 al 20 con un paso de 2.
> Convierte la secuencia de números en una lista y muestra los elementos en la consola.

5.21 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de secuencias de números utilizando range y cómo convertirlas en listas para trabajar con los elementos individualmente. Les ayuda a comprender cómo generar secuencias de números en diferentes rangos.

5.22 Diccionarios

5.22.1 Contenido:

Los diccionarios son estructuras de datos que permiten almacenar pares clave-valor. Aprenderemos cómo crear y trabajar con diccionarios en Python.

5.23 Conceptos Clave:

5.23.1 Diccionarios

Estructuras de datos que almacenan pares clave-valor.

5.23.2 Claves

Son los nombres o etiquetas utilizados para acceder a los valores en el diccionario.

5.23.3 Valores

Son los datos asociados a cada clave en el diccionario.

5.24 Ejemplo:

```
# Creación de un diccionario
persona = {
    "nombre": "Juan",
    "edad": 30,
    "ciudad": "México"
}

# Acceso a valores utilizando claves
nombre = persona["nombre"]
edad = persona["edad"]
```

5.25 Explicación:

En este ejemplo, se crea un diccionario que almacena información de una persona, como nombre, edad y ciudad.

Se accede a los valores del diccionario utilizando las claves correspondientes.

! Actividad Práctica

Crea un diccionario que almacene información de tus libros favoritos, incluyendo título y autor.
Accede a los valores del diccionario utilizando las claves y muestra la información en la consola.

5.26 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de diccionarios y acceder a los valores utilizando las claves. Les ayuda a comprender cómo organizar datos en pares clave-valor y cómo acceder a la información de manera eficiente.

5.27 Booleanos

5.27.1 Contenido

Los booleanos son un tipo de dato que puede tener dos valores: True (verdadero) o False (falso). Aprenderemos cómo trabajar con booleanos en Python y cómo utilizarlos en expresiones lógicas.

5.28 Conceptos Clave:

5.28.1 Booleanos

Tipo de dato que representa valores de verdad (True o False).

Expresiones Lógicas: Combinaciones de valores booleanos utilizando operadores lógicos como and, or y not.

5.29 Ejemplo:

```
# Variables booleanas
es_mayor_de_edad = True
tiene_tarjeta = False
```

```
# Expresiones lógicas
puede_ingresar = es_mayor_de_edad and tiene_tarjeta
```

5.30 Explicación:

En este ejemplo, se utilizan variables booleanas para representar si alguien es mayor de edad y si tiene una tarjeta.

Se utiliza una expresión lógica para evaluar si alguien puede ingresar basado en ambas condiciones.

> **!** Actividad Práctica:
>
> Crea variables booleanas que representen si tienes una mascota y si te gusta el deporte. Utiliza expresiones lógicas para determinar si puedes llevar a tu mascota a un lugar que requiere tu atención durante un partido de tu deporte favorito.

5.31 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de variables booleanas y expresiones lógicas para tomar decisiones basadas en condiciones booleanas. Les ayuda a comprender cómo trabajar con valores de verdad y cómo utilizarlos para evaluar situaciones en el código.

6 Control de Flujo

6.1 Introducción a If

6.1.1 Contenido:

El control de flujo es fundamental en la programación para tomar decisiones basadas en condiciones. Aprenderemos cómo utilizar la estructura if para ejecutar diferentes bloques de código según una condición.

6.2 Conceptos Clave:

6.2.1 Control de Flujo

Manejo de la ejecución del código basado en condiciones.

6.2.2 Estructura if

Permite ejecutar un bloque de código si una condición es verdadera.

6.2.3 Bloque de Código

Conjunto de instrucciones que se ejecutan si la condición es verdadera.

6.3 Ejemplo:

```
edad = 18

if edad >= 18:
    print("Eres mayor de edad.")
```

6.4 Explicación:

En este ejemplo, se utiliza la estructura if para verificar si la variable “edad” es mayor o igual a 18.

Si la condición es verdadera, se ejecuta el bloque de código que muestra un mensaje.

> ! Actividad Práctica:
>
> Crea una variable que represente tu puntuación en un juego.
> Utiliza una estructura if para mostrar un mensaje diferente según si tu puntuación es mayor o igual a 100.

6.5 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la utilización de la estructura if para tomar decisiones basadas en condiciones. Les ayuda a comprender cómo ejecutar diferentes bloques de código según la situación y cómo utilizar el control de flujo en sus programas.

6.6 If y Condicionales

6.6.1 Contenido:

En esta lección, aprenderemos cómo trabajar con múltiples condiciones utilizando la estructura if, elif y else. Esto permite ejecutar diferentes bloques de código según diferentes condiciones.

6.7 Conceptos Clave

6.7.1 Estructura elif

Permite verificar una condición adicional si la condición anterior es falsa.

6.7.2 Estructura else

Define un bloque de código que se ejecuta si todas las condiciones anteriores son falsas.

6.7.3 Anidación de Estructuras if

Es posible anidar múltiples estructuras if para manejar situaciones más complejas.

6.8 Ejemplo:

```
puntaje = 85

if puntaje >= 90:
    print("¡Excelente trabajo!")
elif puntaje >= 70:
    print("Buen trabajo.")
else:
    print("Necesitas mejorar.")
```

6.9 Explicación:

En este ejemplo, se utiliza la estructura if, elif y else para evaluar diferentes rangos de puntajes y mostrar mensajes correspondientes.

> ! Actividad Práctica:
>
> Crea una variable que represente tu calificación en un examen.
> Utiliza una estructura if, elif y else para mostrar mensajes diferentes según la calificación obtenida.

6.10 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la estructura if, elif y else para manejar múltiples condiciones y decisiones en sus programas. Les ayuda a comprender cómo ejecutar diferentes bloques de código en función de los resultados de las pruebas.

6.11 If, elif y else

6.11.1 Contenido:

En esta lección, continuaremos trabajando con la estructura if, elif y else, pero esta vez exploraremos cómo anidar estas estructuras para manejar situaciones más complejas.

6.12 Conceptos Clave:

6.12.1 Anidación de Estructuras

Posibilidad de colocar una estructura if, elif y else dentro de otra.

6.12.2 Jerarquía de Condiciones

Se evalúan las condiciones de manera secuencial y se ejecuta el primer bloque de código correspondiente a una condición verdadera.

6.13 Ejemplo

```
edad = 16
permiso_padres = True

if edad >= 18:
    print("Eres mayor de edad.")
else:
    if permiso_padres:
        print("Eres menor de edad pero tienes permiso de tus padres.")
    else:
        print("Eres menor de edad y no tienes permiso de tus padres.")
```

6.14 Explicación:

En este ejemplo, se anidan estructuras if para manejar diferentes casos basados en la edad y el permiso de los padres.

! Actividad Práctica:

Crea una variable que represente si un usuario está registrado en un sitio web.
Si el usuario está registrado, muestra un mensaje de bienvenida. Si no está registrado, muestra un mensaje que lo invite a registrarse.

6.15 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la anidación de estructuras if, elif y else para manejar situaciones con múltiples condiciones. Les ayuda a comprender cómo trabajar con jerarquías de condiciones y cómo manejar casos más complejos en sus programas.

6.16 And y Or

6.16.1 Contenido:

En esta lección, exploraremos los operadores lógicos and y or, que permiten combinar condiciones para crear expresiones más complejas en las estructuras if, elif y else.

6.17 Conceptos Clave:

6.17.1 Operador and

Retorna True si ambas condiciones son verdaderas.

6.17.2 Operador or

Retorna True si al menos una de las condiciones es verdadera.

6.17.3 Combinación de Condiciones

Los operadores and y or permiten combinar múltiples condiciones en una sola expresión.

6.18 Ejemplo:

```
edad = 20
tiene_permiso = True

if edad >= 18 and tiene_permiso:
    print("Puedes ingresar.")
else:
    print("No puedes ingresar.")
```

6.19 Explicación:

En este ejemplo, se utiliza el operador and para evaluar si la edad es mayor o igual a 18 y si el usuario tiene permiso.

Si ambas condiciones son verdaderas, se permite el ingreso.

> ! Actividad Práctica:
>
> Crea dos variables que representen si un usuario tiene una cuenta premium y si su suscripción está activa.
> Utiliza una estructura if y el operador and para determinar si el usuario tiene acceso premium.

6.20 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la combinación de condiciones utilizando los operadores and y or. Les ayuda a comprender cómo crear expresiones más complejas para tomar decisiones basadas en múltiples condiciones en sus programas.

6.21 Introducción a While

6.21.1 Contenido:

En esta lección, introduciremos la estructura while, que permite ejecutar un bloque de código repetidamente mientras se cumpla una condición.

6.22 Conceptos Clave:

6.22.1 Estructura while

Permite ejecutar un bloque de código mientras una condición sea verdadera.

6.22.2 Condición

La condición se verifica antes de cada iteración. Si es verdadera, se ejecuta el bloque de código.

6.23 Ejemplo:

```
contador = 0

while contador < 5:
    print("Contador:", contador)
    contador += 1
```

6.24 Explicación:

En este ejemplo, se utiliza la estructura while para imprimir el valor del contador mientras sea menor que 5.

Después de cada iteración, se incrementa el contador en 1.

::: {.callout-important} ### Actividad Práctica:

Crea una variable que represente la cantidad de intentos de un usuario para ingresar una contraseña correcta.

Utiliza una estructura while para pedir al usuario que ingrese su contraseña hasta que lo haga correctamente.

6.25 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la estructura while para crear bucles que se ejecutan repetidamente mientras se cumple una condición. Les ayuda a comprender cómo implementar lógica de repetición en sus programas.

6.26 While loop

6.26.1 Contenido:

En esta lección, profundizaremos en el uso de la estructura while para crear bucles que se ejecutan repetidamente mientras se cumpla una condición, y aprenderemos a utilizar la sentencia break para salir de un bucle.

6.27 Conceptos Clave:

6.27.1 Sentencia break:

Se utiliza para salir de un bucle antes de que la condición sea falsa.

6.27.2 Bucles Infinitos:

Si no se maneja adecuadamente, un bucle while puede ejecutarse infinitamente.

6.28 Ejemplo:

```
contador = 0

while True:
    print("Contador:", contador)
    contador += 1
    if contador >= 5:
        break
```

6.29 Explicación:

En este ejemplo, se utiliza un bucle while que se ejecuta infinitamente.

Se utiliza la sentencia break para salir del bucle cuando el contador llega a 5.

! Actividad Práctica:

Crea un bucle while que pida al usuario ingresar un número positivo menor que 10. Utiliza la sentencia break para salir del bucle una vez que el usuario ingrese un número válido.

6.30 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de la sentencia break para controlar la ejecución de un bucle while y evitar bucles infinitos. Les ayuda a comprender cómo manejar situaciones en las que es necesario salir de un bucle antes de que la condición sea falsa.

6.31 While, break y continue

6.31.1 Contenido:

En esta lección, continuaremos explorando cómo trabajar con la estructura while y aprenderemos a utilizar la sentencia continue para saltar a la siguiente iteración del bucle.

6.32 Conceptos Clave:

6.32.1 Sentencia continue

Se utiliza para saltar a la siguiente iteración del bucle sin ejecutar el resto del código en esa iteración.

6.32.2 Saltar Iteraciones

La sentencia continue permite omitir ciertas iteraciones basadas en una condición.

6.33 Ejemplo

```
contador = 0

while contador < 5:
    contador += 1
    if contador == 3:
        continue
    print("Contador:", contador)
```

6.34 Explicación:

En este ejemplo, se utiliza un bucle while para imprimir el valor del contador.

Se utiliza la sentencia continue para omitir la iteración cuando el contador es igual a 3.

::: {.callout-important} ### Actividad Práctica:

Crea un bucle while que imprima los números del 1 al 10, pero omite la impresión del número 5.

Utiliza la sentencia continue para lograr esto.

```
## Explicación de la Actividad

Esta actividad permite a los participantes practicar el uso de la sentencia continue para omi

## For Loop

### Contenido:

En esta lección, aprenderemos sobre el bucle for, que se utiliza para iterar sobre secuencias

## Conceptos Clave:

### Bucle for

Se utiliza para recorrer elementos en una secuencia uno por uno.

### Iteración
```

```
En cada iteración del bucle, se ejecuta un bloque de código con un valor diferente de la secu

### range() con Bucles for

Se puede utilizar la función range() para generar una secuencia de números y recorrerla con u

## Ejemplo:

```python
frutas = ["manzana", "banana", "naranja"]

for fruta in frutas:
    print("Me gusta", fruta)
```

6.35 Explicación:

En este ejemplo, se utiliza un bucle for para recorrer una lista de frutas.

En cada iteración, se asigna el valor actual de la lista a la variable fruta.

> **!** Actividad Práctica:
>
> Crea una lista de colores.
> Utiliza un bucle for para imprimir cada color en la lista precedido por la palabra “Color:”.

6.36 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso del bucle for para iterar sobre elementos en una secuencia. Les ayuda a comprender cómo trabajar con bucles para recorrer listas y otros tipos de secuencias en sus programas.
```

7 Funciones

7.1 Introducción a Funciones

7.1.1 Contenido:

En esta lección, exploraremos el concepto de funciones en Python. Aprenderemos cómo definir funciones, pasar argumentos y cómo retornar valores.

7.2 Conceptos Clave:

7.2.1 Funciones.

Bloques de código reutilizables que realizan una tarea específica.

7.2.2 Definición de Funciones

Se utiliza la palabra clave def para definir una función.

7.2.3 Argumentos

Valores que se pasan a una función para que trabaje con ellos.

7.3 Retorno de Valores

Una función puede retornar un valor utilizando la palabra clave return.

7.4 Ejemplo:

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

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

7.5 Explicación:

En este ejemplo, se define una función llamada saludar que toma un argumento nombre.

La función retorna un saludo personalizado utilizando el argumento.

El resultado se asigna a la variable mensaje y se muestra en la consola.

> ❗ Actividad Práctica:
>
> Crea una función llamada calcular_cuadrado que reciba un número como argumento y retorne el cuadrado de ese número.
> Utiliza la función para calcular el cuadrado de un número y mostrarlo en la consola.

7.6 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación y uso de funciones en Python. Les ayuda a comprender cómo definir funciones, pasar argumentos y cómo trabajar con valores retornados por las funciones.

7.7 Recursividad

7.7.1 Contenido:

En esta lección, exploraremos el concepto de recursividad, que es cuando una función se llama a sí misma para resolver un problema. Aprenderemos cómo implementar funciones recursivas y cuándo es apropiado usarlas.

7.8 Conceptos Clave:

7.8.1 Recursividad

Técnica en la que una función se llama a sí misma para resolver un problema.

7.8.2 Caso Base

Condición que indica cuándo la recursión debe detenerse.

7.8.3 Caso Recursivo

Cómo se divide el problema en partes más pequeñas en cada llamada recursiva.

7.9 Ejemplo:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

resultado = factorial(5)
print(resultado)
```

7.10 Explicación:

En este ejemplo, se define una función recursiva llamada factorial para calcular el factorial de un número.

La función utiliza un caso base (cuando n es 0) y un caso recursivo (llamando a la función con n - 1).

> ! Actividad Práctica:
>
> Crea una función recursiva llamada potencia que calcule la potencia de un número base elevado a un exponente.

> Utiliza la función para calcular 2^3 y muestra el resultado en la consola.

7.11 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la implementación de funciones recursivas y comprender cómo se dividen los problemas en partes más pequeñas para resolverlos. Les ayuda a comprender cómo aplicar la recursividad de manera efectiva en la solución de problemas.

8 Objetos, Clases y Herencia

8.1 Introducción

8.2 Contenido:

En esta lección, exploraremos el concepto de programación orientada a objetos (POO). Aprenderemos sobre objetos, clases y cómo la POO nos permite organizar y estructurar nuestro código de manera más eficiente.

8.3 Conceptos Clave:

8.3.1 Programación Orientada a Objetos (POO)

Paradigma de programación que se basa en el uso de objetos y clases.

8.3.2 Objetos

Instancias de clases que representan entidades en el mundo real.

8.3.3 Clases

Plantillas o moldes que definen la estructura y el comportamiento de los objetos.

8.3.4 Atributos

Características o propiedades de un objeto.

8.3.5 Métodos

Funciones que definen el comportamiento de un objeto.

8.4 Ejemplo:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")

persona1 = Persona("Juan", 25)
persona1.saludar()
```

8.5 Explicación:

En este ejemplo, se define una clase llamada Persona con un constructor (**init**) que inicializa atributos.

La clase tiene un método llamado saludar que muestra un mensaje con el nombre y edad del objeto.

> ! Actividad Práctica:
>
> Crea una clase llamada Libro con atributos titulo y autor, y un método mostrar_info que imprima los atributos.
> Crea una instancia de la clase Libro y llama al método mostrar_info.

8.6 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la definición de clases y la creación de objetos. Les ayuda a comprender cómo la POO nos permite modelar entidades y organizar el código de manera más estructurada y eficiente.

8.7 Objetos y Clases

8.7.1 Contenido:

En esta lección, continuaremos explorando los conceptos de objetos y clases en la programación orientada a objetos. Aprenderemos cómo crear múltiples objetos a partir de una misma clase y cómo trabajar con sus atributos y métodos.

8.8 Conceptos Clave:

8.8.1 Instancias de Clase

Cuando se crea un objeto a partir de una clase, se crea una instancia de esa clase.

8.8.2 Atributos de Instancia:

Características específicas de un objeto que se almacenan como variables en la instancia.

8.8.3 Métodos de Instancia

Funciones definidas en la clase que operan en los atributos de la instancia.

8.9 Ejemplo:

```
class Perro:
    def __init__(self, nombre, raza):
        self.nombre = nombre
        self.raza = raza

    def ladrar(self):
        print(f"{self.nombre} está ladrando.")

perro1 = Perro("Max", "Labrador")
perro2 = Perro("Buddy", "Chihuahua")

perro1.ladrar()
perro2.ladrar()
```

8.10 Explicación:

En este ejemplo, se define una clase Perro con un constructor y un método ladrar.

Se crean dos objetos (perro1 y perro2) a partir de la misma clase y se les asignan diferentes valores para sus atributos.

Los métodos de instancia son llamados en cada objeto para realizar acciones específicas.

> ! Actividad Práctica:
>
> Crea una clase Rectangulo con atributos ancho y alto, y un método calcular_area que calcule y retorne el área del rectángulo.
> Crea dos instancias de la clase Rectangulo con diferentes valores de ancho y alto, y llama al método calcular_area en cada una.

8.11 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de instancias de clase y trabajar con sus atributos y métodos. Les ayuda a comprender cómo cada objeto puede tener valores diferentes para sus atributos y cómo ejecutar acciones específicas en cada objeto.

8.12 Métodos

8.12.1 Contenido:

En esta lección, profundizaremos en el concepto de métodos en la programación orientada a objetos. Aprenderemos cómo definir y utilizar métodos en una clase, y cómo acceder a los atributos de instancia dentro de los métodos.

8.13 Conceptos Clave:

8.13.1 Métodos de Clase

Funciones definidas dentro de una clase que operan en los atributos de instancia.

8.13.2 Acceso a Atributos

Dentro de un método, se puede acceder a los atributos de instancia utilizando self.atributo.

8.14 Ejemplo:

```
class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def calcular_area(self):
        area = 3.14 * self.radio ** 2
        return area

circulo1 = Circulo(5)
area_circulo = circulo1.calcular_area()
print("Área del círculo:", area_circulo)
```

8.15 Explicación:

En este ejemplo, se define una clase Circulo con un constructor y un método calcular_area.

Dentro del método, se accede al atributo de instancia radio utilizando self.radio para calcular el área.

> ! Actividad Práctica:
>
> Crea una clase Triangulo con atributos base y altura, y un método calcular_area que calcule y retorne el área del triángulo.
> Crea una instancia de la clase Triangulo y llama al método calcular_area para calcular el área.

8.16 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la definición y uso de métodos en una clase. Les ayuda a comprender cómo trabajar con atributos de instancia dentro de los métodos y cómo implementar lógica específica para cada objeto.

8.17 Self, Eliminar Propiedades y Objetos

8.17.1 Contenido:

En esta lección, aprenderemos más sobre el uso de self en los métodos de clase. También exploraremos cómo eliminar atributos de instancia y objetos en Python.

8.18 Conceptos Clave:

8.18.1 self

Referencia al objeto actual en un método de clase.

8.18.2 Eliminar Atributos

Se puede usar la palabra clave del para eliminar un atributo de instancia.

8.18.3 Eliminar Objetos

Se utiliza la función del para eliminar un objeto y liberar memoria.

8.19 Ejemplo:

```
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def mostrar_info(self):
        print(f"Coche {self.marca} {self.modelo}")

coche1 = Coche("Toyota", "Corolla")
coche1.mostrar_info()

# Eliminar el atributo 'modelo'
del coche1.modelo
```

```
# Intentar acceder al atributo eliminado generará un error
# print(coche1.modelo)
```

8.20 Explicación:

En este ejemplo, se define una clase Coche con un constructor y un método mostrar_info.

Se crea una instancia coche1 y se muestra su información. Luego, se elimina el atributo modelo utilizando del.

> ! Actividad Práctica:
>
> Crea una clase Estudiante con atributos nombre y edad, y un método mostrar_info para mostrar la información del estudiante.
> Crea una instancia de la clase Estudiante y llama al método mostrar_info.
> Utiliza del para eliminar el atributo nombre de la instancia y verifica el resultado.

8.21 Explicación de la Actividad:

Esta actividad permite a los participantes practicar el uso de self en los métodos de clase y cómo eliminar atributos de instancia. Les ayuda a comprender cómo trabajar con objetos y atributos, y cómo gestionar la memoria en Python.

8.22 Herencia

8.23 Contenido:

En esta lección, exploraremos el concepto de herencia en la programación orientada a objetos. Aprenderemos cómo crear clases que heredan atributos y métodos de una clase base.

8.24 Conceptos Clave:

8.24.1 Herencia

Mecanismo que permite que una clase herede atributos y métodos de otra clase base.

8.24.2 Clase Padre (o Base)

La clase de la que se heredan atributos y métodos.

8.24.3 Clase Hija (o Derivada)

La clase que hereda de la clase base.

8.25 Ejemplo:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        print(f"{self.nombre} saluda")

class Perro(Animal):
    def ladrar(self):
        print(f"{self.nombre} está ladrando")

perro1 = Perro("Buddy")
perro1.saludar()
perro1.ladrar()
```

8.25.1 Explicación:

En este ejemplo, se define una clase base Animal con un constructor y un método saludar.

Se define una clase derivada Perro que hereda de Animal y agrega un método adicional ladrar.

Se crea una instancia perro1 de la clase Perro y se llama a sus métodos.

> ❗ Actividad Práctica:
>
> Crea una clase Figura con un atributo color y un método mostrar_color para mostrar el color.
> Crea una clase derivada Circulo que herede de Figura y agrega un atributo radio y un método calcular_area para calcular el área del círculo.
> Crea una instancia de la clase Circulo, establece su color y calcula el área.

8.26 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de clases derivadas y la herencia de atributos y métodos. Les ayuda a comprender cómo utilizar la herencia para reutilizar código y extender funcionalidades en las clases derivadas.

9 Módulos

9.1 Introducción

9.1.1 Contenido:

En esta lección, exploraremos cómo trabajar con módulos en Python. Aprenderemos cómo dividir nuestro código en módulos reutilizables y cómo importarlos en otros programas.

9.2 Conceptos Clave:

9.2.1 Módulos

Archivos que contienen código Python y se utilizan para organizar y reutilizar funciones, clases y variables.

9.2.2 Importar Módulos

Se utiliza la palabra clave import para cargar un módulo en un programa.

9.2.3 Usar Funciones y Clases

Después de importar un módulo, sus funciones y clases pueden ser utilizadas como si estuvieran definidas en el mismo archivo.

9.3 Ejemplo:

```
# En el archivo calculadora.py
def suma(a, b):
    return a + b
```

```python
# En otro archivo
import calculadora

resultado = calculadora.suma(3, 5)
print("Resultado:", resultado)
```

9.4 Explicación:

En este ejemplo, se define una función suma en el módulo calculadora.py.

En otro archivo, se importa el módulo calculadora utilizando import y se utiliza la función suma del módulo.

> ! Actividad Práctica:
>
> Crea un módulo llamado matematicas con una función multiplicacion que multiplique dos números.
> Importa el módulo en otro archivo y utiliza la función multiplicacion para calcular el producto de dos números.

9.5 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación y uso de módulos en Python. Les ayuda a comprender cómo organizar su código en módulos reutilizables y cómo importar funciones y clases desde otros archivos.

9.6 Creando Nuestro Primer Módulo

9.6.1 Contenido:

En esta lección, aprenderemos a crear nuestro propio módulo en Python. Crearemos un módulo que contenga funciones y clases para realizar operaciones matemáticas básicas.

9.7 Pasos para Crear un Módulo:

Crea un archivo de Python con la extensión .py.

Define funciones y clases en el archivo.

Guarda el archivo en una ubicación accesible.

9.8 Ejemplo:

```
# En el archivo operaciones.py
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b

class Calculadora:
    def multiplicacion(self, a, b):
        return a * b
```

9.9 Explicación:

En este ejemplo, se crea un módulo llamado operaciones.py.

Se define una función suma y una función resta, junto con una clase Calculadora que tiene un método multiplicacion.

> ! Actividad Práctica:
>
> Crea un módulo llamado geometria con funciones para calcular el área de un círculo y el perímetro de un cuadrado.
> En otro archivo, importa el módulo geometria y utiliza las funciones para realizar cálculos geométricos.

9.10 Explicación de la Actividad:

Esta actividad permite a los participantes practicar la creación de módulos con funciones y clases. Les ayuda a comprender cómo organizar diferentes funcionalidades en módulos separados y cómo importar esas funcionalidades en otros archivos.