Python 2023

Diego Saavedra

Sep 12, 2023

Table of contents

1 Bienvenida

¡Bienvenidos al Curso Completo de Python, analizaremos desde los 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!

Part I

Unidad 1: Introducción a Python

2 Introducción general a la Programación

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.

2.1 Conceptos Clave

2.1.1 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.

2.1.2 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.

2.1.3 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.

2.1.4 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.

2.2 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.

2.3 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.

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

2.4 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

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.

3.1 Conceptos Clave

3.1.1 Python

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

3.1.2 Interprete

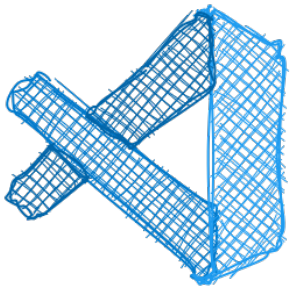
Python es un lenguaje interpretado, lo que significa que se ejecuta línea por línea en tiempo real.

3.1.3 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.

3.2 Ejemplo

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



3.3 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:

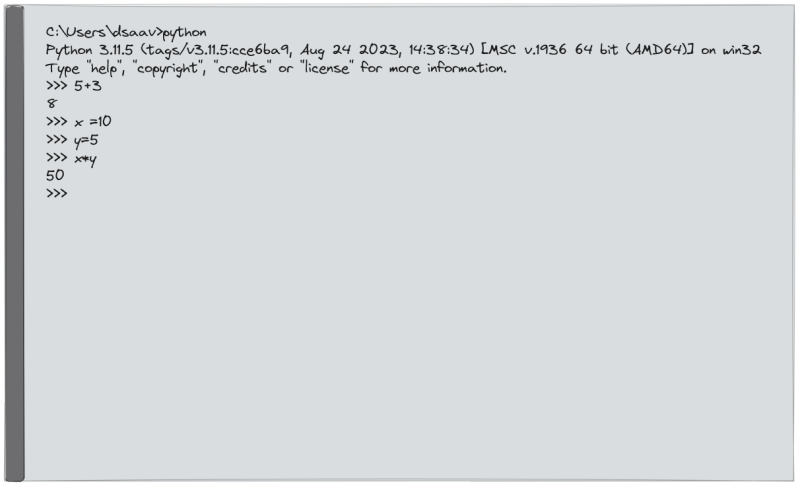
```
print("Python se ha instalado correctamente.")
```

3.4 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 Uso del REPL, PEP 8 y el Zen de Python



4.0.1 El REPL (Read-Eval-Print Loop).

Definición y Propósito del REPL.

- El REPL (Read-Eval-Print Loop) es una herramienta interactiva que permite ejecutar código Python de forma inmediata y ver los resultados de las operaciones en tiempo real. Es una excelente manera de probar pequeños fragmentos de código, experimentar y depurar sin necesidad de escribir un programa completo.

Uso Básico del REPL

Para iniciar el REPL, simplemente abre una terminal o línea de comandos y escribe python o python3 (dependiendo de tu instalación) seguido de Enter. Esto te llevará al entorno interactivo de Python.

4.0.2 Ejemplos de Interacción con el REPL

```
# Ejemplo 1: Realizar cálculos simples
>>> 5 + 3
8

# Ejemplo 2: Definir variables y realizar operaciones
>>> x = 10
>>> y = 5
```

```
>>> x * y
50

# Ejemplo 3: Trabajar con cadenas de texto
>>> mensaje = "Hola, mundo!"
>>> mensaje.upper()
'HOLA, MUNDO!'

# Ejemplo 4: Importar módulos y usar funciones
>>> import math
>>> math.sqrt(16)
4.0
```

4.0.3 PEP 8: Guía de Estilo de Python.

¿Qué es PEP 8 y Por Qué es Importante?

PEP 8 (Python Enhancement Proposal 8) es una guía de estilo que establece convenciones para escribir código Python legible y consistente. La adopción de PEP 8 es importante porque facilita la colaboración en proyectos, mejora la legibilidad del código y ayuda a mantener una base de código ordenada y coherente.

Convenciones de Nombres

PEP 8 establece reglas para nombrar variables, funciones, clases y módulos en Python. Algunas convenciones clave incluyen:

- Las variables y funciones deben usar minúsculas y palabras separadas por guiones bajos (snake_case).
- Las clases deben usar CamelCase (con la primera letra en mayúscula). Los módulos deben tener nombres cortos y en minúsculas.

Reglas de Formato y Estilo

PEP 8 también define reglas de formato, como el uso de espacios en lugar de tabulaciones, la longitud máxima de línea y la organización de importaciones.

Herramientas para Verificar el Cumplimiento de PEP 8

Existen herramientas como flake8 y complementos para editores de código que pueden analizar el código en busca de posibles violaciones de PEP 8 y proporcionar sugerencias de corrección. 2.4.3. El Zen de Python

4.0.4 Introducción al Zen de Python (PEP 20).

El Zen de Python es un conjunto de principios y filosofía de diseño que guían el desarrollo de Python. Estos principios se pueden acceder desde el intérprete de Python utilizando el siguiente comando:

```
import this
```

Los principios del Zen de Python proporcionan orientación sobre cómo escribir código Python de manera clara y elegante.

Principios y Filosofía de Diseño de Python

Algunos de los principios más destacados del Zen de Python incluyen:

- **La legibilidad cuenta:** El código debe ser legible para los humanos, ya que se lee con más frecuencia de lo que se escribe.
- **Explícito es mejor que implícito:** El código debe ser claro y no dejar lugar a ambigüedades.
- **La simplicidad vence a la complejidad:** Debe preferirse la simplicidad en el diseño y la implementación.
- **Los errores nunca deben pasar en silencio:** Los errores deben manejarse adecuadamente y, si es posible, informar de manera explícita.

4.0.5 Ejercicios Prácticos

- **Ejercicio 1:** Uso del REPL para Realizar Cálculos Simples

① Abre el REPL de Python.

- **Ejercicio 2:** Verificación de Cumplimiento de PEP 8 en Código Python
- Escribe un pequeño programa en Python que incluya variables, funciones y comentarios.

- Utiliza la herramienta flake8 o un complemento de tu editor de código para verificar si tu código cumple con las reglas de PEP 8.
- Corrige cualquier violación de PEP 8 y vuelve a verificar el código.
- **Ejercicio 3:** Exploración y Reflexión sobre los Principios del Zen de Python
- Ejecuta el comando import this en el REPL para acceder a los principios del Zen de Python.
- Lee y reflexiona sobre cada uno de los principios.
- Escribe un breve párrafo sobre cómo un principio específico del Zen de Python puede aplicarse al desarrollo de software.

Tip

Actividad Práctica

- Desarrolla un pequeño programa Python que siga las pautas de PEP 8 y refleje los principios del Zen de Python en su diseño y estilo de codificación. Asegúrate de que el código sea legible y cumpla con las convenciones de nombres y formato de PEP 8.
- Esta subunidad proporciona a los estudiantes una comprensión más profunda de las herramientas y las convenciones de estilo que se utilizan en la programación en Python. Además, les ayuda a reflexionar sobre la filosofía de diseño de Python y cómo aplicarla en la práctica.

4.0.6 Explicación

Esta actividad te invita a desarrollar un pequeño programa en Python que siga las pautas de PEP 8 y refleje los principios del Zen de Python en su diseño y estilo de codificación. La importancia de esta tarea radica en aprender a escribir código que sea limpio, legible y siga las convenciones de la comunidad de Python.

- **Cumplir con PEP 8**: PEP 8 es el estándar de estilo de código para Python, y seguirlo es una práctica recomendada en la comunidad de programadores. Tu programa debe seguir las convenciones de formato, nombres de variables, estructura de código, entre otros aspectos que se describen en PEP 8.
- **Reflejar el Zen de Python**: El Zen de Python es una colección de principios filosóficos que guían el diseño del lenguaje Python. Algunos de estos principios incluyen la legibilidad del código, la simplicidad y la importancia de los casos especiales. Tu programa debe reflejar estos principios en su diseño y estilo de codificación.
- **Legibilidad y Comentarios**: Asegúrate de que tu código sea legible para otras personas. Usa nombres de variables descriptivos, agrega comentarios explicativos cuando sea necesario y sigue las mejores prácticas para hacer que tu código sea fácil de entender.

- **Aplicación Práctica**: Esta actividad te brinda la oportunidad de aplicar los conceptos de estilo de código y filosofía de diseño de Python en un proyecto real. Esto es importante ya que en la programación colaborativa, otros desarrolladores deben poder entender y trabajar con tu código de manera eficiente.

Al completar esta actividad, habrás mejorado tus habilidades en la escritura de código Python de alta calidad y te habrás familiarizado con las convenciones y filosofía de diseño de Python. Recuerda que escribir código limpio es una habilidad esencial para cualquier programador.

5 Entornos de Desarrollo

Part II

Unidad 2: Introducción a la Programación con Python

6 Identación y Comentarios

6.1 Identación

6.2 Conceptos Clave

6.2.1 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.

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")
```

Actividad Práctica

1. Escribe un programa que solicite al usuario su edad y muestre un mensaje según si es mayor de 18 años o no.

Posible solución

Resumen:

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.

```
# Programa que solicita la edad y muestra un mensaje
edad = int(input("Ingrese su edad: "))

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

```
else:
    print("Eres menor de edad.")
```

¿Qué hicimos?

① Se solicita la edad al usuario y se almacena en la variable edad.

6.3 Comentarios

6.4 Conceptos Clave

6.4.1 Comentarios

- Son notas en el código que no se ejecutan y se utilizan para explicar el propósito y funcionamiento de partes del programa.
- Comentarios de una línea: Se crean con el símbolo "#" y abarcan una sola línea.
- Comentarios de múltiples líneas: Se crean entre triple comillas (" " " o ' ' ') y pueden abarcar múltiples líneas.

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
```

> 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.

Posible solución

```
# Este programa calcula el área de un triángulo
# solicitando la base y la altura al usuario.

# Solicitar la base y almacenarla en la variable 'base'
base = float(input("Ingrese la base del triángulo: "))

# Solicitar la altura y almacenarla en la variable 'altura'
```

```
altura = float(input("Ingrese la altura del triángulo: "))

# Calcular el área del triángulo
area = 0.5 * base * altura

# Mostrar el resultado
print(f"El área del triángulo es: {area}")
```

6.5 ¿Qué aprendimos?

En este tema, aprendimos la importancia de la identación en Python para estructurar nuestro código correctamente. La identación nos permite definir bloques de código, como en los condicionales, de manera clara y legible.

También aprendimos cómo agregar comentarios en Python para documentar nuestro código. Los comentarios son esenciales para explicar el propósito y el funcionamiento de las partes del programa y facilitan la colaboración entre programadores.

7 Variables y Variables Múltiples

7.1 Variables

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

7.1.1 Conceptos Clave

Variables

- Nombres que representan ubicaciones de memoria donde se almacenan datos.

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).

Ejemplo:

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

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

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

7.2 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. Múltiples Variables

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

7.2.1 Conceptos Clave

Asignación Múltiple

- Permite asignar valores a varias variables en una línea.

Desempaquetado de Valores

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

Intercambio de Valores

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

Ejemplo:

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

7.2.2 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.

Actividad Práctica

1. Crea una lista con los nombres de tus tres colores favoritos.
2. Utiliza la asignación múltiple para asignar los valores de la lista a tres variables individuales.

7.2.3 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.

8 Concatenación

8.1 Conceptos Clave

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

Operador +: Se utiliza para concatenar cadenas de texto.

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

8.2 Ejemplo

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

8.2.1 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.

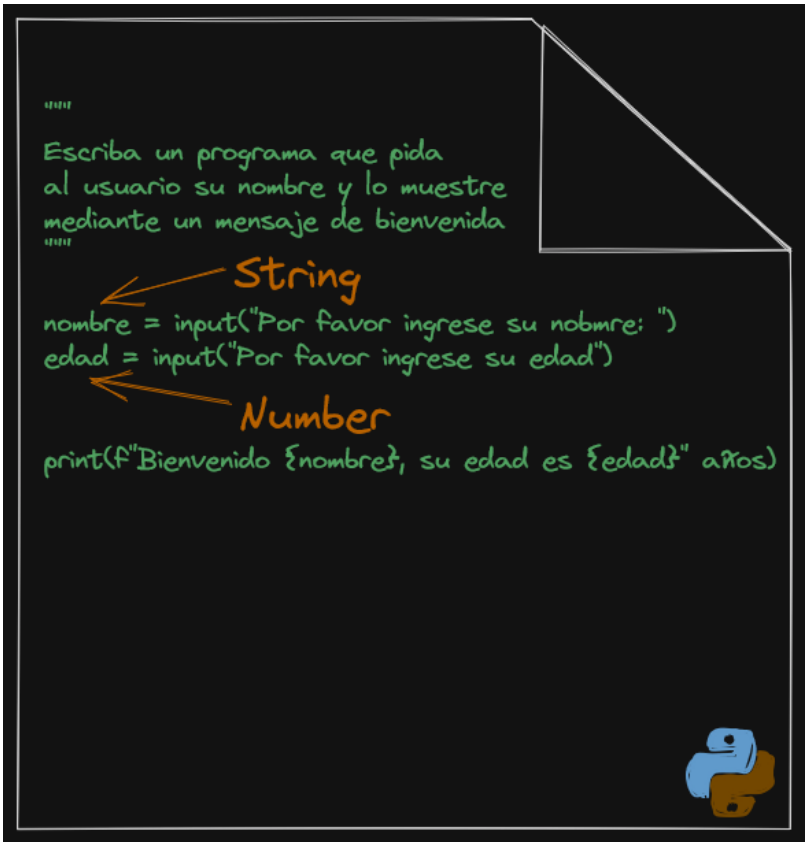
8.2.2 Explicación de la Actividad:

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.

Part III

Unidad 3: Tipos de Datos

9 String y Números



9.1 Conceptos Clave

String

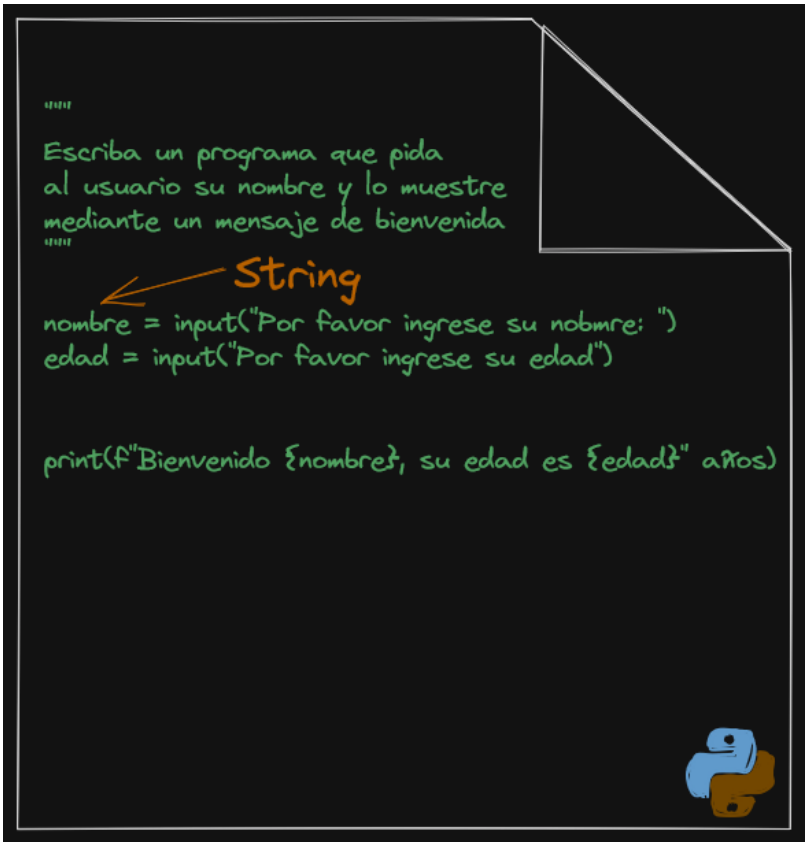
Un string es una secuencia de caracteres alfanuméricos. Se pueden definir utilizando comillas simples o dobles.

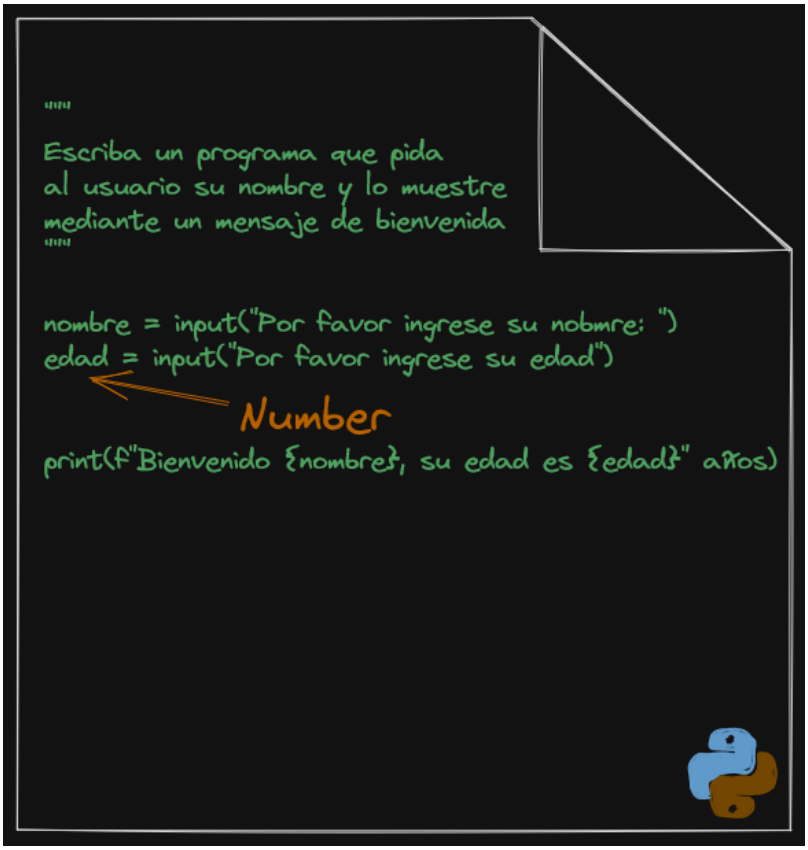
Números Enteros (int)

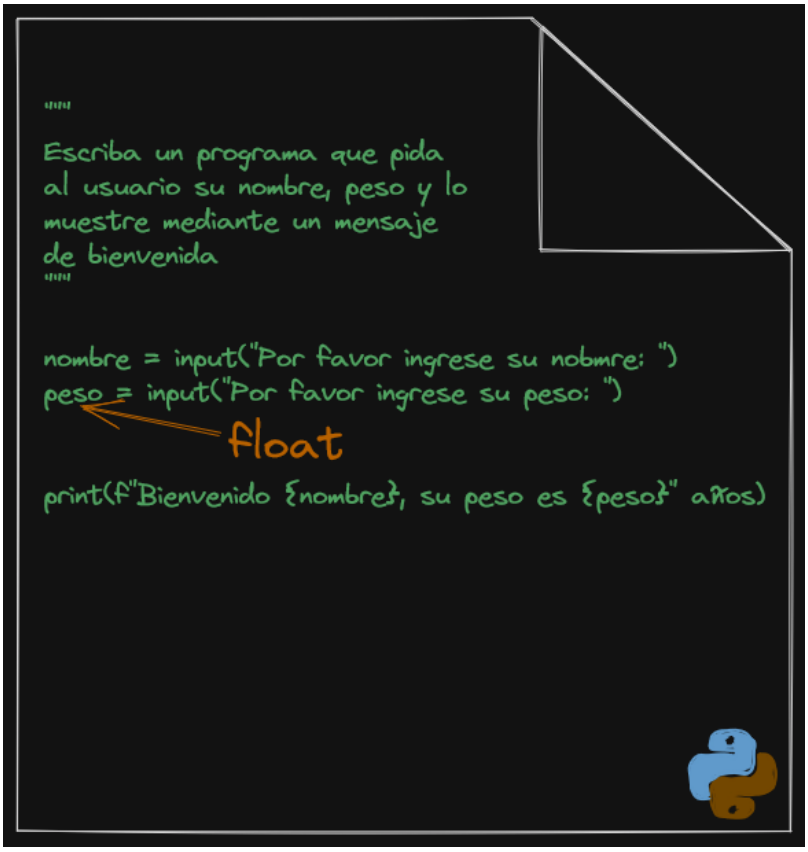
Los números enteros representan valores numéricos enteros, ya sean positivos o negativos.

Números de Punto Flotante (float)

Los números de punto flotante representan valores numéricos con decimales.


"""
Escriba un programa que pida
al usuario su nombre y lo muestre
mediante un mensaje de bienvenida
"""
String
nombre = input("Por favor ingrese su nobmre: ")
edad = input("Por favor ingrese su edad")
print(f"Bienvenido {nombre}, su edad es {edad}" años)


"""
Escriba un programa que pida
al usuario su nombre y lo muestre
mediante un mensaje de bienvenida
"""
nombre = input("Por favor ingrese su nobmre: ")
edad = input("Por favor ingrese su edad")
Number
print(f"Bienvenido {nombre}, su edad es {edad}" años)



9.2 Ejemplo

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

# Números
edad = 25
saldo = 1500.75
```

9.2.1 Explicación:

En este ejemplo, se crean variables que almacenan strings y números. Los strings se definen utilizando comillas simples o dobles, y los números enteros y de punto flotante se asignan directamente a variables.

Actividad Práctica

1. Crea una variable con el título de tu canción favorita.
2. Asigna tu edad a una variable y tu altura a otra variable.
3. Combina las variables para crear un mensaje personalizado.

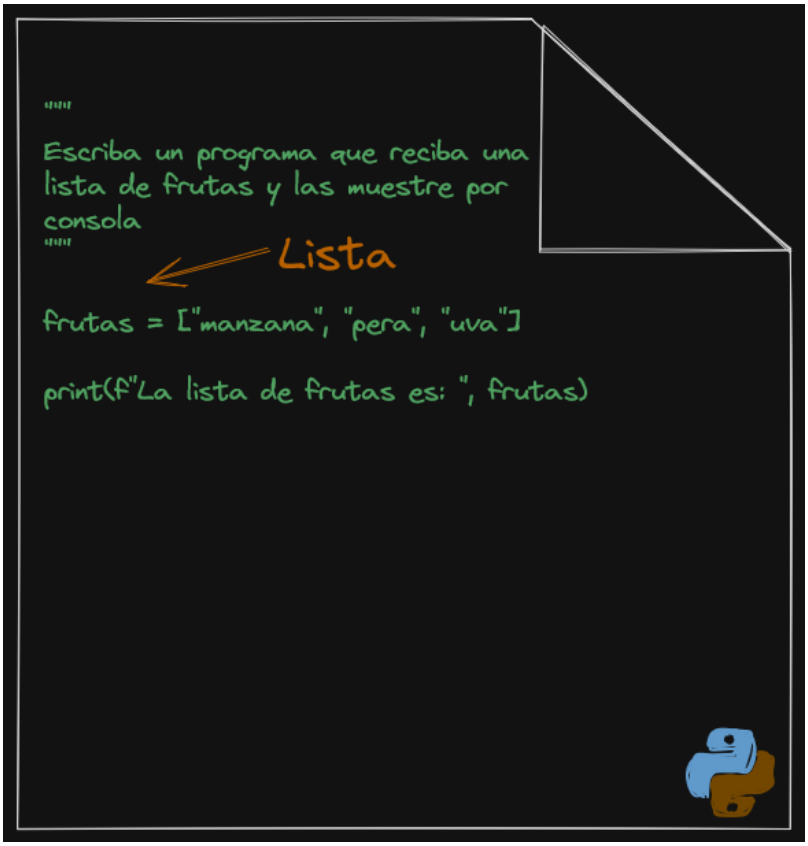
9.2.2 Explicación:

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.

9.3 ¿Qué Aprendimos?

En esta lección, aprendimos sobre los dos tipos de datos fundamentales en Python: strings y números. Aprendimos cómo crear y trabajar con strings utilizando comillas simples o dobles. También exploramos dos tipos numéricos clave: números enteros (int) y números de punto flotante (float). Estos conceptos son esenciales para manejar información en Python y forman la base de muchos programas y aplicaciones.

10 Listas y Tuplas



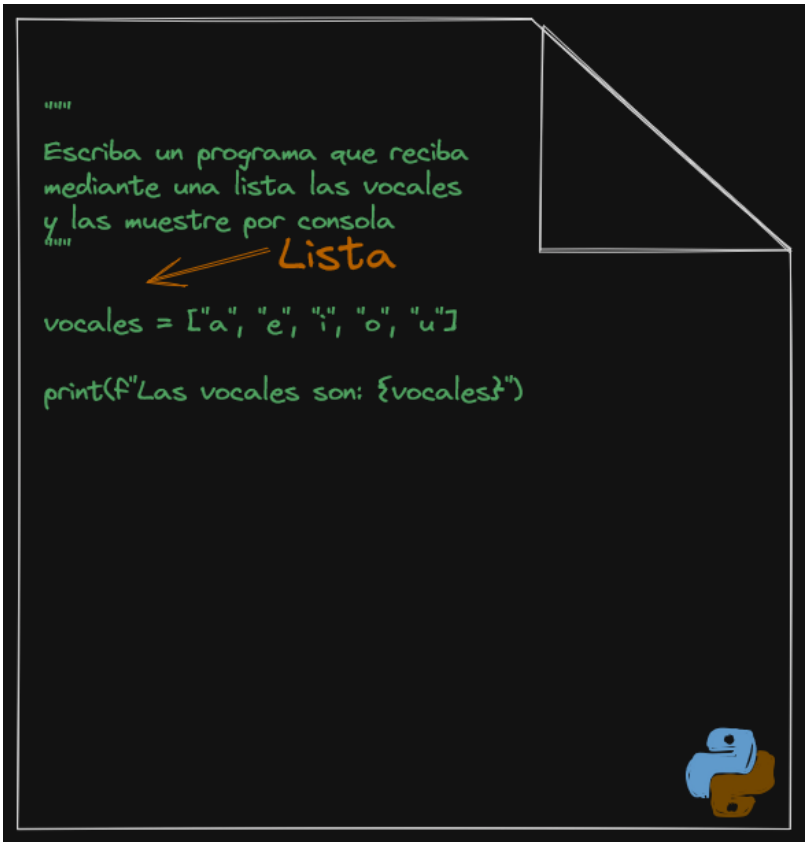
10.1 Conceptos Clave

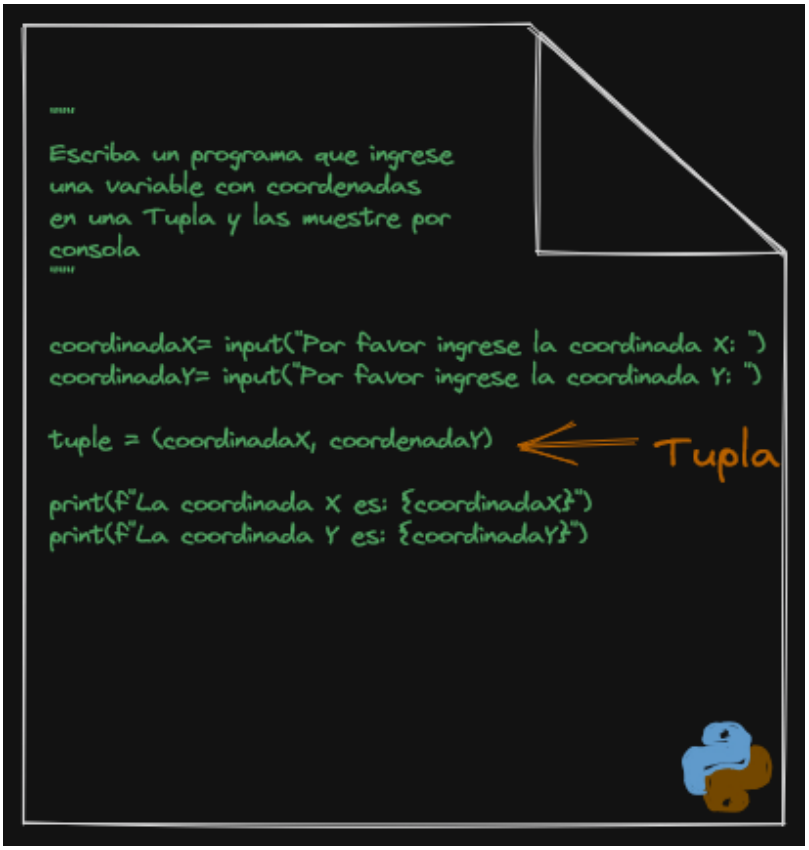
10.1.1 Listas

Las listas son secuencias ordenadas de elementos que pueden ser de diferentes tipos. Permiten almacenar varios elementos en una sola variable.

10.1.2 Tuplas

Las tuplas son similares a las listas, pero son inmutables, lo que significa que no se pueden modificar después de ser creadas.


"""
Escriba un programa que reciba
mediante una lista las vocales
y las muestre por consola
"""
Lista
vocales = ["a", "e", "i", "o", "u"]
print(f"Las vocales son: {vocales}")


"""
Escriba un programa que ingrese
una variable con coordenadas
en una Tupla y las muestre por
consola
"""
coordinadaX= input("Por favor ingrese la coordinada X: ")
coordinadaY= input("Por favor ingrese la coordinada Y: ")
tuple = (coordinadaX, coordenadaY)
Tupla
print(f"La coordinada X es: {coordinadaX}")
print(f"La coordinada Y es: {coordinadaY}")

10.2 Ejemplo

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

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

10.2.1 Explicación:

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

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

Actividad Práctica (Listas):

1. Crea una lista con los nombres de tus tres películas favoritas.
2. Accede al segundo elemento de la lista e imprímelo en la consola.

10.2.2 Explicación:

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.

Actividad Práctica (Tuplas):

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

10.2.3 Explicación:

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.

10.3 ¿Qué Aprendimos?

En esta lección, aprendimos sobre dos tipos de estructuras de datos en Python: listas y tuplas.

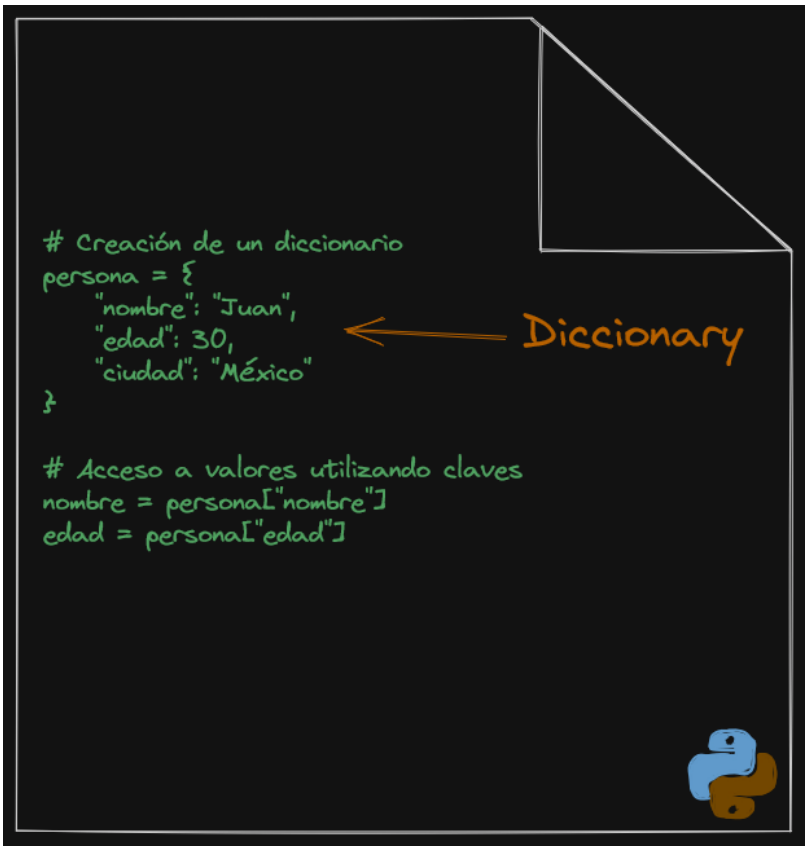
Listas: Son secuencias ordenadas de elementos que pueden modificarse. Se accede a los elementos utilizando índices.

Tuplas: Son similares a las listas, pero son inmutables, lo que significa que no pueden modificarse después de su creación. También se accede a los elementos utilizando índices.

Estas estructuras nos permiten almacenar y organizar datos de manera eficiente en Python, y la elección entre listas y tuplas depende de si necesitamos datos modificables o inmutables en nuestros programas.

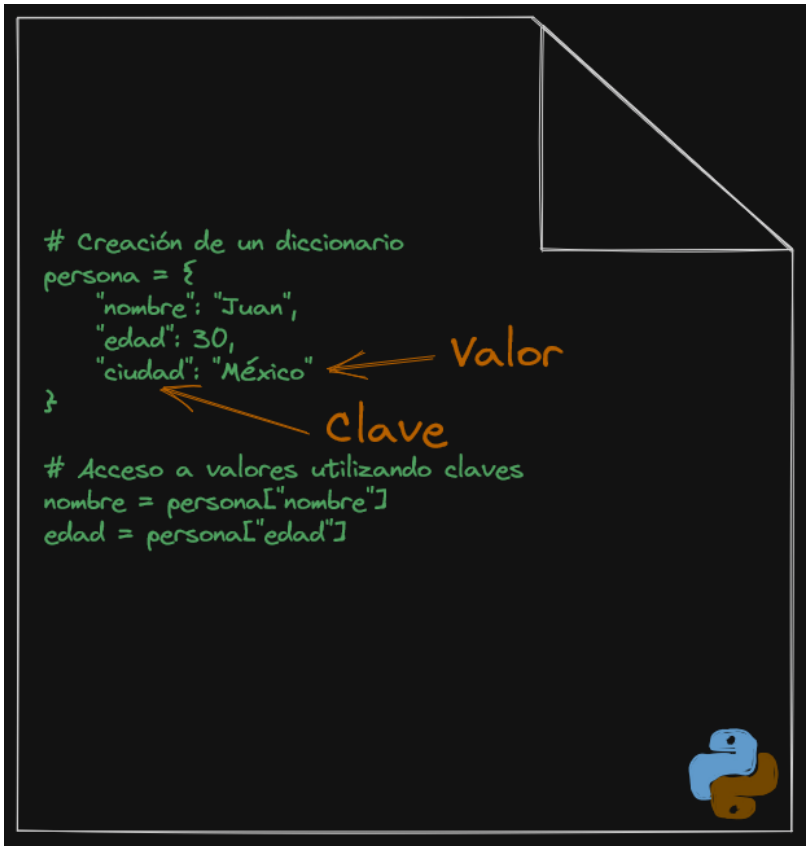
11 Diccionarios y Booleanos

11.1 Diccionarios



11.1.1 Conceptos Clave

11.1.1.1 Diccionarios



Los diccionarios son estructuras de datos que almacenan pares clave-valor.

11.1.1.2 Claves

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

11.1.1.3 Valores

Son los datos asociados a cada clave en el diccionario.

11.1.2 Ejemplo

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

```
}

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

11.1.3 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
>
> 1. Crea un diccionario que almacene información de tus libros favoritos, incluyendo título y autor.
> 2. Accede a los valores del diccionario utilizando las claves y muestra la información en la consola.

11.1.4 Explicación:

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.

11.2 Booleanos

11.3 Conceptos Clave

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. Ejemplo

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

# Expresiones lógicas
puede_ingresar = es_mayor_de_edad and tiene_tarjeta
```

11.3.1 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.

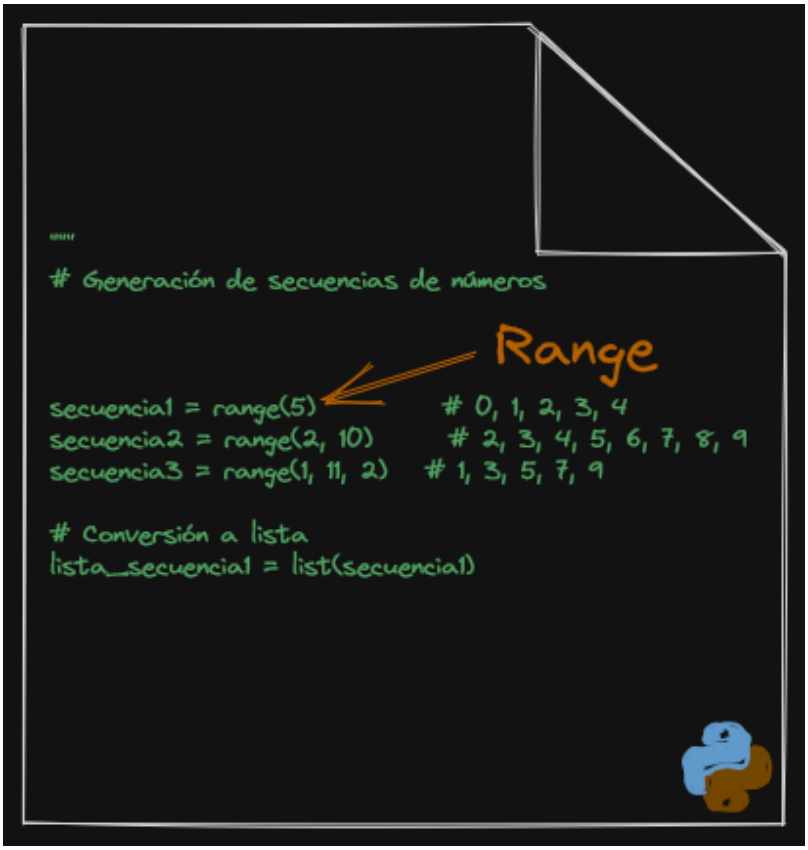
Actividad Práctica

1. Crea variables booleanas que representen si tienes una mascota y si te gusta el deporte.
2. 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.

11.3.2 Explicación:

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.

12 Range



12.1 Conceptos Clave

12.1.1 range

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

12.1.2 Parámetros de range

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

12.1.3 Conversión a Listas

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

12.2 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)
```

12.2.1 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

1. Crea una secuencia de números del 10 al 20 con un paso de 2.
2. Convierte la secuencia de números en una lista y muestra los elementos en la consola.

12.2.2 Explicación:

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.

Part IV

Unidad 4: Control de Flujo

13 Introducción a If

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.

13.1 Conceptos Clave

13.1.1 Control de Flujo

Manejo de la ejecución del código basado en condiciones.

13.1.2 Estructura if

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

13.1.3 Bloque de Código

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

13.2 Ejemplo

```
edad = 18

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

13.2.1 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

1. Crea una variable que represente tu puntuación en un juego.
2. Utiliza una estructura if para mostrar un mensaje diferente según si tu puntuación es mayor o igual a 100.

13.2.2 Explicación:

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.

13.3 ¿Qué aprendimos?

En este tema, aprendimos los conceptos fundamentales del control de flujo en programación y cómo utilizar la estructura if para ejecutar código condicionalmente. Ahora somos capaces de tomar decisiones en nuestros programas basadas en condiciones específicas.

14 If y Condicionales

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.

14.1 Conceptos Clave

14.1.1 Estructura elif

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

14.1.2 Estructura else

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

14.1.3 Anidación de Estructuras if

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

14.2 Ejemplo

```
puntaje = 85

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

14.2.1 Explicación:

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

Actividad Práctica

1. Crea una variable que represente tu calificación en un examen.
2. Utiliza una estructura if, elif y else para mostrar mensajes diferentes según la calificación obtenida.

14.2.2 Explicación:

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.

14.3 ¿Qué aprendimos?

En este tema, aprendimos a utilizar la estructura if, elif y else para manejar múltiples condiciones y ejecutar código basado en resultados específicos. También comprendimos cómo anidar estructuras if para manejar situaciones más complejas en la programación.

15 If, elif y else

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

15.1 Conceptos Clave

15.1.1 Estructura elif

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

15.1.2 Estructura else

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

15.1.3 Anidación de Estructuras if

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

15.2 Ejemplo

```
puntaje = 85

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

15.2.1 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.

15.2.2 Explicación:

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.

15.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la estructura if, elif y else para tomar decisiones basadas en múltiples condiciones. Esto nos permite ejecutar diferentes bloques de código según diferentes situaciones. También exploramos la anidación de estructuras if, lo que nos permite manejar situaciones aún más complejas en la programación.

16 And y Or

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.

16.1 Conceptos Clave:

16.1.1 Operador and

Retorna True si ambas condiciones son verdaderas.

16.1.2 Operador or

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

16.1.3 Combinación de Condiciones

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

16.2 Ejemplo:

```
edad = 20
tiene_permiso = True

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

16.2.1 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.

16.2.2 Explicación:

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.

16.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar los operadores lógicos and y or para combinar condiciones y crear expresiones más complejas en nuestras estructuras de control de flujo. Estos operadores son útiles cuando necesitamos tomar decisiones basadas en múltiples condiciones en nuestros programas.

17 Introducción a While.

17.1 Introducción a While

En esta lección, comenzaremos a explorar la estructura de control de flujo while, que nos permite crear bucles que se ejecutan repetidamente mientras se cumple una condición.

17.2 Conceptos Clave

17.2.1 Bucle While

Un bucle que se ejecuta mientras una condición sea verdadera.

17.2.2 Condición

La expresión que se evalúa para determinar si el bucle debe continuar ejecutándose.

17.2.3 Bloque de Código

El conjunto de instrucciones que se ejecutan dentro del bucle while.

17.3 Ejemplo

```
contador = 0

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

17.3.1 Explicación:

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

17.3.2 Explicación:

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.

17.4 ¿Qué aprendimos?

En esta lección, aprendimos los conceptos clave de la estructura de control de flujo while. Descubrimos cómo crear bucles que se ejecutan mientras se cumple una condición y cómo utilizarlos en situaciones prácticas en la programación.

18 While loop.

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.

18.1 Conceptos Clave:

18.1.1 Sentencia break:

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

18.1.2 Bucles Infinitos:

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

18.2 Ejemplo:

```
contador = 0

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

18.2.1 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:

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

18.2.2 Explicación:

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.

18.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la estructura while para crear bucles en Python que se ejecutan mientras se cumpla una condición. También aprendimos a utilizar la sentencia break para salir de un bucle antes de que la condición sea falsa. Los bucles while son útiles cuando necesitamos ejecutar un bloque de código repetidamente hasta que se cumpla una condición específica.

19 While, break y continue.

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.

19.1 Conceptos Clave:

19.1.1 Sentencia continue

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

19.1.2 Saltar Iteraciones

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

19.2 Ejemplo

```
contador = 0

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

19.2.1 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.

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.

19.2.2 Explicación:

Esta actividad permite a los participantes practicar el uso de la sentencia continue para omitir iteraciones específicas en un bucle while. Les ayuda a comprender cómo controlar la ejecución de un bucle y realizar acciones selectivas en cada iteración.

19.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo utilizar la sentencia continue en un bucle while para saltar a la siguiente iteración sin ejecutar el resto del código en esa iteración. Esto es útil cuando queremos omitir ciertas iteraciones basadas en una condición específica. El control preciso de las iteraciones en un bucle puede ser esencial para realizar tareas específicas en un programa.

20 For Loop

En esta lección, aprenderemos sobre la estructura de control de flujo for loop en Python. El bucle for nos permite recorrer elementos de una secuencia, como una lista o una cadena de texto.

20.1 Conceptos Clave

20.1.1 Bucle For

Un bucle que itera a través de una secuencia de elementos y ejecuta un bloque de código para cada elemento.

20.1.2 Iteración

Cada ejecución del bloque de código en un bucle for se llama iteración.

20.1.3 Elemento de la Secuencia

Los elementos individuales en la secuencia que se está recorriendo.

20.2 Ejemplo

```
frutas = ["manzana", "banana", "cereza"]

for fruta in frutas:
    print(fruta)
```

20.2.1 Explicación:

En este ejemplo, utilizamos un bucle for para iterar a través de la lista frutas e imprimimos cada fruta en la consola.

> Actividad Práctica
>
> Crea una lista de números del 1 al 5 y utiliza un bucle for para imprimir el cuadrado de cada número.

20.2.2 Explicación:

Esta actividad te permite practicar cómo usar un bucle for para recorrer una secuencia de números y realizar operaciones en cada elemento. Los bucles for son muy útiles para procesar datos en una variedad de situaciones de programación.

20.3 ¿Qué aprendimos?

En esta lección, aprendimos cómo usar un bucle for para iterar a través de una secuencia de elementos en Python. Comprendimos los conceptos clave relacionados con los bucles for y cómo aplicarlos en la escritura de código.

Part V

Unidad 5: Funciones y Recursividad

21 Introducción a Funciones

En esta lección, exploraremos el concepto de funciones en Python. Las funciones son bloques de código reutilizables que realizan tareas específicas. Aprenderemos cómo definir y utilizar funciones en nuestros programas.

21.1 Conceptos Clave

21.1.1 Función

Un bloque de código reutilizable que realiza una tarea específica cuando se llama.

21.1.2 Definición de Función

Crear una función especificando su nombre, parámetros y cuerpo.

21.1.3 Llamada de Función

Ejecutar una función para que realice su tarea específica.

21.2 Ejemplo

```
# Definición de una función
def saludar(nombre):
    print("¡Hola, " + nombre + "!")

# Llamada de función
saludar("Juan")
```

21.2.1 Explicación:

En este ejemplo, definimos una función llamada saludar que toma un parámetro nombre e imprime un saludo personalizado. Luego, llamamos a esta función con el nombre “Juan”.

> **Tip**
>
> Actividad Práctica
> Crea una función llamada calcular_area_rectangulo que tome dos parámetros: largo y ancho. La función debe calcular y devolver el área de un rectángulo. Luego, llama a la función con valores diferentes para largo y ancho y muestra el resultado.

21.2.2 Explicación:

Esta actividad te permite practicar cómo definir funciones con parámetros y cómo utilizarlas para realizar cálculos específicos. Las funciones son una parte fundamental de la programación, ya que permiten organizar y reutilizar el código de manera efectiva.

21.3 ¿Qué aprendimos?

En esta lección, aprendimos qué son las funciones en Python y cómo definirlas y usarlas en nuestros programas. Comprendimos los conceptos clave relacionados con las funciones y cómo aplicarlos en la escritura de código.