**Introducción**

Python es uno de los lenguajes de programación más populares hoy en día. Fue creado en 1989 por Guido van Rossum y lanzado por primera vez en 1991. Es un lenguaje de programación conocido por su simplicidad y legibilidad, lo cual facilita su aprendizaje y usabilidad por parte tanto de profesionales principiantes como avanzados.

En este documento, abordaremos los fundamentos básicos de Python, dando una mirada profunda a su sintaxis, estructuras de datos, y más, proporcionando una base sólida para adentrarte en el mundo de la programación con Python.

1. **Sintaxis Básica**

Nada es posible en programación sin la correcta sintaxis, es por ello que en esta primera sección nos adentraremos en los fundamentos de la sintaxis de Python, abarcando temas como la identación, variables, tipos de datos y operadores para conocer cómo estructurar correctamente las combinaciones en este lenguaje.

* **Identación**

Python utiliza la identación para definir bloques de código, lo cual se refiere a los espacios al inicio de una línea de código. A diferencia de otros lenguajes que usan llaves para definir bloques de código, Python usa la identacion para delimitar la estructura y establecer bloques de código, por lo cual es considerado un lenguaje visualmente más limpio y organizado. Te invito a verlo en el siguiente ejemplo:

if 5 > 2:

print ("Cinco es mayor que dos")

En el ejemplo anterior, el bloque de código dentro del if está identado con dos espacios, indicando que pertenece al if.

Una buena práctica para que el código sea fácil de leer y comprender, es mantener una identación consistente en tu código, lo que generalmente se logra usando un editor de código que automáticamente aplique una identación uniforme.

* **Variables**

Las variables son contenedores donde podemos almacenar valores dentro de una estructura de código. En Python, las variables se crean cuando se les asigna un valor por primera vez. Dado que Python es dinámicamente tipado, no es necesario declarar el tipo de una variable al crearla. Vamos a verlo en un ejemplo:

x = 5

y = "Hola, mundo"

En el ejemplo anterior, x es una variable de tipo entero y mientras que y es una variable de tipo cadena (string).

Los nombres de las variables en Python pueden contener letras, números y guiones bajos, sin embargo siempre hay que recordar que los nombres de las variables no pueden comenzar con un número. Para que el código sea legible, se recomienda que los nombres sean descriptivos.

* **Tipos de datos**

Un tipo de dato hace referencia a la clasificación de los valores que pueden estar contenidos en una variable. La programación con Python soporta varios tipos de datos, incluyendo:

* Integers (enteros): números sin punto decimal.
* Floats (flotantes): números con punto decimal.
* Strings (cadenas): secuencia de caracteres encerrados entre comillas.
* Booleans (booleanos): valores verdadero (True) o falso (False).

Ejemplo:

x = 5 # Entero

y = 5.5 # Flotante

z = "Hola" # Cadena

w = True # Booleano

Python permite cambiar el tipo de una variable a través de un proceso conocido como casting. Por ejemplo, podemos cambiar un flotante a un entero usando la función int() tal como se muestra en la siguiente secuencia de código:

x = 5.5

x = int(x)

print(x) # Salida: 5

* **Operadores**

Python ofrece una amplia gama de operadores que permiten realizar los cálculos que se necesiten durante la programación. Es por ello que podemos identificar operadores aritméticos para realizar operaciones matemáticas y operadores de comparación para comparar valores.

* Operadores aritméticos

Los operadores aritméticos incluyen operaciones básicas como suma (+), resta (-), multiplicación (\*), y división (/). Si consideramos a x = 5 y

y = 2, en el siguiente ejemplo podemos ver dichas operaciones:

print(x + y) # Suma

print(x - y) # Resta

print(x \* y) # Multiplicación

print(x / y) # División

* Operadores de comparación

Los operadores de comparación se usan para comparar dos valores, es por ello que incluyen:

Igual a (==)

Diferente de (!=)

Mayor que (>)

Menor que (<)

Mayor o igual a (>=)

Menor o igual a (<=).

Si consideramos a x = 5 y y = 2, en el siguiente ejemplo podemos ver aplicadas dichas operaciones:

print(x == y) # Igual a

print(x != y) # Diferente de

print(x > y) # Mayor que

1. **Estructuras de Datos**

Las estructuras de datos son una forma de organizar y almacenar datos. En este sentido, Python ofrece varias estructuras de datos integradas, como listas, tuplas, diccionarios y conjuntos. Vamos a ver a continuación a que hace referencia cada una y ejemplos para poder comprenderlas:

* **Listas**

Las listas son una de las estructuras de datos más versátiles en Python, las cuales sirven para almacenar colecciones de ítems en una sola variable. Las listas están ordenadas, lo que significa que los ıtems tienen un orden definido, y a la vez son mutables dado que podemos cambiar, agregar, y remover ítems después de que la lista está definida. En el siguiente caso podemos ver una lista definida:

mylist = ["manzana", "banana", "cherry"]

print(mylist)

Podemos acceder a los ítems de una lista refiriéndonos al número de índice del ítem. Los índices en Python comienzan en 0. Veamos esto en un ejemplo:

mylist = ["manzana", "banana", "cherry"]

print(mylist[1]) # Salida: banana

Los elementos de una lista pueden modificarse refiriéndonos al número de índice. En este caso, se haría de la siguiente manera:

mylist = ["manzana", "banana", "cherry"]

mylist[1] = "arándano"

print(mylist) # Salida: [’manzana’, ’arándano’, ’cherry’]

* **Tuplas**

Las tuplas son similares a las listas, pero son inmutables. Esto significa que no podemos cambiar, agregar, o remover ítems una vez que la tupla está definida.

Ejemplo:

mytuple = ("manzana", "banana", "cherry")

print(mytuple)

Al igual que las listas, podemos acceder a los ítems de una tupla utilizando índices.

Ejemplo:

mytuple = ("manzana", "banana", "cherry")

print(mytuple[1]) # Salida: banana

A pesar de que las tuplas son inmutables, es posible crear una nueva tupla con contenido modificado tomando porciones de otras tuplas. Veamos cómo se podría hacer con un ejemplo:

mytuple = ("manzana", "banana", "cherry")

mynewtuple = mytuple[:1] + ("arándano",) + mytuple[2:]

print(mynewtuple) # Salida: (’manzana’, ’arándano’, ’cherry’)

* **Diccionarios**

Los diccionarios son estructuras de datos que permiten almacenar pares de clave-valor. Las claves deben ser únicas dentro de un diccionario, mientras que los valores pueden ser de cualquier tipo y hasta repetirse. Te muestro un ejemplo:

mydict = {

"nombre": "Juan",

"edad": 30

}

print(mydict)

Podemos acceder a los valores de un diccionario utilizando las claves correspondientes.

Ejemplo:

mydict = {

"nombre": "Juan",

"edad": 30

}

print(mydict["nombre"]) # Salida: Juan

Los valores de un diccionario pueden modificarse refiriéndose a las claves correspondientes. Veamos un ejemplo:

mydict = {

"nombre": "Juan",

"edad": 30

}

mydict["edad"] = 31

print(mydict) # Salida: {’nombre’: ’Juan’, ’edad’: 31}

* **Conjuntos**

Los conjuntos son colecciones no ordenadas y sin índices de ítems únicos. Los conjuntos son útiles para almacenar elementos sin un orden particular y para realizar operaciones de conjuntos, como unión, intersección y diferencia.

Ejemplo:

myset = {"manzana", "banana", "cherry"}

print(myset)

Podemos agregar y remover elementos de un conjunto usando los métodos add() y remove(), respectivamente. Te lo muestro aplicado en un caso:

myset = {"manzana", "banana", "cherry"}

myset.add("arándano")

myset.remove("banana")

print(myset) # Salida: {’manzana’, ’cherry’, ’arándano’}

Python también soporta operaciones de conjuntos tradicionales, como unión, intersección y diferencia.

Ejemplo:

set1 = {"a", "b", "c"}

set2 = {"d", "e", "f", "a"}

# Unión de conjuntos

print(set1.union(set2))

# Intersección de conjuntos

print(set1.intersection(set2))

# Diferencia de conjuntos

print(set1.difference(set2))

1. **Control de Flujo**

El control de flujo permite dirigir el flujo de ejecución del programa mediante estructuras condicionales y bucles. En esta sección, exploraremos las estructuras condicionales (if-elif-else) y los bucles (for y while) en Python.

* **Condicionales (if-elif-else)**

Las estructuras condicionales permiten ejecutar diferentes bloques de código dependiendo de ciertas condiciones. Empecemos con un ejemplo:

edad = 18

if edad >= 18:

print("Mayor de edad")

else:

print("Menor de edad")

* Elif

El ”elif” es una abreviatura de ”else if” y permite verificar múltiples expresiones para determinar si son verdaderas.

Ejemplo:

edad = 18

if edad > 18:

print("Mayor de edad")

elif edad == 18:

print("Justo 18 años")

else:

print("Menor de edad")

* **Bucles (for, while)**

Los bucles permiten ejecutar un bloque de código varias veces. Python ofrece dos tipos de bucles: for y while.

* Bucle for

El bucle for se utiliza para iterar sobre una secuencia, que puede ser una lista, una tupla, un diccionario, un conjunto o una cadena de caracteres.

Ejemplo:

frutas = ["manzana", "banana", "cherry"]

for fruta in frutas:

print(fruta)

* Bucle while

El bucle while permite ejecutar un conjunto de declaraciones mientras una condición sea verdadera.

Ejemplo:

i = 1

while i < 6:

print(i)

i += 1

1. **Funciones**

Las funciones son bloques de código que están diseñados para hacer un trabajo específico. Una vez que se escribe el código para una función se puede reutilizar en cualquier lugar de un programa. En esta sección, aprenderemos cómo definir y llamar funciones en Python, incluyendo funciones lambda.

* **Definición de funciones**

Para definir una función en Python, usamos la palabra clave def seguida del nombre de la función y una lista de parámetros entre paréntesis. Veamos un ejemplo:

def saludo(nombre):

print(f"Hola, {nombre}")

saludo("Javier")

* Parámetros y argumentos

Cuando definimos una función, podemos especificar los parámetros que aceptará. Al llamar a la función, proporcionamos los valores para estos parámetros, que se conocen como argumentos.

Ejemplo:

def suma(a, b):

return a + b

resultado = suma(5, 3)

print(resultado) # Salida: 8

* **Funciones lambda**

Las funciones lambda son funciones anónimas que pueden tener cualquier número de argumentos pero solo una expresión. Se utilizan para crear funciones pequeñas y simples on-the-fly.

Ejemplo:

potencia = lambda x, y: x \*\* y

print(potencia(2, 3)) # Salida: 8

1. **Módulos**

Los módulos en Python son archivos que contienen código Python. Un módulo puede definir funciones, clases y variables que se pueden reutilizar en otros archivos Python. En esta sección, veremos cómo importar módulos y utilizar el código que contienen.

* **Importación de módulos**

Para utilizar un módulo en Python, primero debemos importarlo usando la instrucción import. Una vez importado, podemos acceder a las funciones y variables definidas en ese módulo.

Ejemplo:

import math

print(math.sqrt(16)) # Salida: 4.0

* Importación selectiva

También, podemos elegir importar solo algunas funciones o variables específicas de un módulo usando la sintaxis from ... import ....

Ejemplo:

from math import sqrt

print(sqrt(16)) # Salida: 4.0

* Alias

Podemos darle un alias a un módulo al importarlo, lo que permite referirse a este con un nombre diferente. Veamos un ejemplo:

import math as m

print(m.sqrt(16)) # Salida: 4.0

1. **Manejo de Archivos**

Python facilita la lectura y escritura de archivos, permitiendo así manipular datos almacenados en disco desde un programa Python. En el siguiente apartado veremos cómo leer y escribir archivos de texto y también cómo manejar errores que pueden ocurrir al realizar estas operaciones.

* **Lectura y escritura básica**

Podemos leer y escribir archivos en Python usando las funciones built-in open(), read(), write(), y close().

Para escribir contenido a un archivo, usamos el método write() de la forma en que se puede ver en el siguiente ejemplo:

with open(’miarchivo.txt’, ’w’) as f:

f.write(’Hola Mundo’)

En el código anterior, utilizamos la instrucción with para abrir el archivo en modo de escritura (’w’) y escribir una cadena dentro del archivo. El uso de with asegura que el archivo se cierre automáticamente al final del bloque.

Para leer el contenido de un archivo, usamos el método read() y el cual se aplica de la manera en que podemos verlo en el siguiente ejemplo:

with open(’miarchivo.txt’, ’r’) as f:

contenido = f.read()

print(contenido) # Salida: Hola Mundo

**Cierre**

A través de este repaso, hemos explorado los conceptos fundamentales de Python, el cual es considerado un lenguaje de programación versátil y poderoso. A partir de la comprensión sólida de estos conceptos, estás listo para comenzar a desarrollar tus propios programas en Python.