INTRODUCCIÓN A PYTHON

Contenido

Sintaxis en Python

Python es conocido por su sintaxis sencilla y fácil de leer, lo que lo convierte en un lenguaje ideal para programadores de todos los niveles.

La sintaxis de Python se diseñó con el objetivo de ser intuitiva y promover un código limpio y legible, siguiendo el principio de que “lo bello es mejor que lo feo” del Zen de Python. Esto se refleja en la forma en que se escribe y estructura el código en Python, facilitando la comprensión y el mantenimiento del mismo.

Espacios en blanco significativos

A diferencia de otros lenguajes, Python utiliza los espacios en blanco (indentaciones) para definir la estructura del código en lugar de llaves o palabras clave específicas.

Esto significa que el nivel de indentación de una línea de código indica su relación con las líneas anteriores, estableciendo así bloques de código. Por ejemplo, todos los statements que pertenecen a una misma función deben tener el mismo nivel de indentación.

Comentarios

Los comentarios en Python se realizan usando el símbolo # para comentarios de una sola línea, y triple comillas dobles """ para comentarios de múltiples líneas o docstrings, los cuales son especialmente útiles para documentar el propósito de funciones y clases.

Variables y tipos de datos

En Python, las variables son como cajas mágicas donde puedes guardar casi cualquier cosa que desees, sin tener que etiquetarlas con tipos específicos.

Esta flexibilidad proviene de ser un lenguaje de tipado dinámico, lo que significa que Python entiende qué tipo de objeto estás almacenando al momento que lo almacenas.

Esto es porque Python infiere el tipo de dato, permitiendo una sintaxis más ágil y menos verbosa. Por ejemplo, para asignar un valor a una variable, simplemente utilizas el signo de igual (=) sin necesidad de especificar qué va a almacenar:

```
mi_variable = 10
```

Estructuras de control

Las estructuras de control en Python incluyen condicionales lógicos (if, elif, else) y bucles (for, while). Estas estructuras permiten dirigir el flujo de ejecución del programa en función de condiciones específicas o repetir un bloque de código múltiples veces.

Estas estructuras proveen formas simplificadas de escribir ciertos fragmentos del código y generar instrucciones que puedan ser ejecutadas dependiendo de ciertas condiciones y un número controlado de veces.

Tipos de datos

En Python, los tipos de datos fundamentales definen la categoría de valores que una variable puede almacenar y cómo el intérprete maneja dicha variable. Comprender estos tipos es crucial para manipular datos de manera efectiva en Python. Veamos los más comunes:

Números

Python admite varios tipos numéricos, incluidos enteros (int), números de punto flotante (float), y números complejos (complex) que pueden combinarse mediante operadores. Los enteros pueden ser de cualquier longitud, los de punto flotante tienen precisión doble, y los complejos incluyen una parte real y una imaginaria.

```
entero = 10

flotante = 10.5

complejo = 3 + 5j
```

Cadenas de texto (Strings)

Las cadenas de texto en Python se definen entre comillas simples (') o dobles ("), y pueden contener caracteres alfanuméricos y símbolos. Python también soporta cadenas multilínea, que se definen con triple comillas (''' o """).

```
cadena_simple = 'Hola, Mundo'

cadena_multilinea = """Esto es una cadena

multilínea en Python."""
```

Booleanos

Los booleanos representan dos valores: Verdadero (True) y Falso (False). Son muy utilizados en expresiones condicionales y bucles.

```
verdadero = True

falso = False

verdadero and falso # Dará False

verdadero or falso # Dará True
```

Listas

Las listas son estructuras de datos que permiten almacenar una colección ordenada y mutable de elementos. Los elementos pueden ser de diferentes tipos, incluyendo otra lista.

```
pythonCopy code

mi_lista = [1, 2.5, 'Python', [3, 4]]
```

Tuplas

Las tuplas son similares a las listas, pero son inmutables. Una vez creada una tupla, no se pueden modificar sus elementos. Es un tipo más ligero ya que su manipulación es mínima.

```
mi_tupla = (1, 2.5, 'Python')
```

Diccionarios

Los diccionarios almacenan pares de clave-valor, siendo una estructura mutable. Las claves deben ser únicas y pueden ser de cualquier tipo inmutable.

```
mi_diccionario = {'nombre': 'Juan', 'edad': 30, 'lenguajes': ['Python', 'JavaScript']}
```

Conjuntos

Los conjuntos son colecciones desordenadas de elementos únicos. Son útiles para realizar operaciones de conjunto como uniones, intersecciones, y diferencias.

```
mi_conjunto = {1, 2, 3, 4, 5}
```

Cada tipo de dato en Python está diseñado para ser utilizado en diferentes escenarios, proporcionando una gran flexibilidad y eficiencia en el manejo de datos. Conociendo estos tipos, los desarrolladores pueden elegir la estructura más adecuada para sus necesidades específicas, optimizando así sus programas.

Gestión de variables

La gestión de variables en Python es un aspecto fundamental que permite a los programadores almacenar, modificar y recuperar datos durante la ejecución de un programa. Python simplifica esta gestión mediante un enfoque dinámico y flexible, que se ajusta a las necesidades de diversos tipos de aplicaciones. Aquí exploraremos cómo se declaran, utilizan y gestionan las variables en Python.

Declaración y asignación

En Python, las variables se crean en el momento en que se les asigna un valor por primera vez. No es necesario declararlas explícitamente con un tipo específico, ya que el tipo de dato se infiere del valor asignado. Esto hace que el código sea más limpio y fácil de escribir y entender.

```
codex = 10       # Entero

y = 3.14      # Flotante

nombre = "Ana"  # Cadena de texto
```

Tipado dinámico

Python es un lenguaje de tipado dinámico, lo que significa que el tipo de una variable puede cambiar durante la ejecución del programa. Esto proporciona una gran flexibilidad,

pero también requiere que el programador sea consciente de los tipos de datos con los que está trabajando para evitar errores.

```
numero = 42    # Aquí 'numero' es un entero

numero = "cuarenta y dos"  # Ahora 'numero' se convierte en una cadena de texto
```

Nombres de variables

Los nombres de las variables en Python pueden contener letras, números y el carácter de subrayado (_), pero no pueden comenzar con un número. Además, Python distingue entre mayúsculas y minúsculas, lo que significa que variable, Variable y VARIABLE serían tres variables diferentes.

Ámbito de las variables

El ámbito de una variable determina la parte del programa en la que la variable es accesible. Python tiene dos ámbitos principales: local y global. Las variables definidas dentro de una función tienen un ámbito local, mientras que las definidas fuera de cualquier función son globales.

```
def mi_funcion():
  var_local = "Esto es local"

var_global = "Esto es global"
```

Palabras clave global y nonlocal

Para modificar una variable global dentro de una función, se debe usar la palabra clave global. Similarmente, nonlocal permite modificar variables de ámbitos superiores en funciones anidadas.

```
contador = 0

def incrementar():
  global contador
  contador += 1
```

La gestión eficiente de variables es crucial para el desarrollo de la lógica de un programa en Python. Entender cómo declarar, asignar y manejar el ámbito de las variables permite a los desarrolladores escribir código más limpio, eficiente y fácil de mantener.

Estructuras de control

Las estructuras de control en Python permiten dirigir el flujo de ejecución de un programa. Estas estructuras son esenciales para tomar decisiones en el código, repetir operaciones y controlar el proceso de ejecución en función de distintas condiciones. Python ofrece varias estructuras de control, cada una adaptada a necesidades específicas.

Condicionales: if, elif, else

La declaración if permite ejecutar un bloque de código si una condición específica es verdadera. elif y else se utilizan para comprobar múltiples condiciones y definir un curso de acción para cada una.

```
edad = 18

if edad < 18:
    print("Menor de edad")
elif edad >= 18:
    print("Mayor de edad")
else:
    print("Este código nunca se ejecutará")
```

Bucles: for y while

Los bucles permiten ejecutar un bloque de código repetidamente. Python proporciona dos tipos de bucles: for y while. El bucle for se utiliza para iterar sobre una secuencia (como una lista, tupla, diccionario, conjunto o cadena). El bucle while se ejecuta mientras una condición sea verdadera.

Bucle for

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

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

Bucle while

```
contador = 0

while contador < 3:
    print("Dentro del bucle")
    contador += 1
```

Comprensiones de listas

La comprensión de listas es una de las características más interesantes de Python que ofrece una manera concisa de crear colecciones de elementos.

Consisten en una expresión seguida de un bucle for dentro de corchetes, que además no sólo permite poblar dicha lista, sino que permiten filtrar elementos de otra lista u operar sobre los elementos de manera eficiente.

```
cuadrados = [x**2 for x in range(10)]
```

Por ejemplo, esta comprensión de lista creará una lista de los primeros 10 cuadrados ([1, 4, 9, 16...100]).

Control de flujo: break, continue, pass

- break termina el bucle más cercano.

- continue omite el resto del código dentro del bucle y continúa con la siguiente iteración.
- pass se utiliza como una declaración de relleno o de lugar; no tiene efecto.

```
for num in range(5):
    if num == 3:
        break    # Sale del bucle
    print(num)
```

Estas estructuras de control son los bloques básicos que permiten a los desarrolladores de Python escribir código flexible y eficiente. Al utilizar de forma inteligente estas estructuras, se pueden resolver problemas complejos de manera efectiva y crear programas más dinámicos y interactivos.

Funciones: Maximizando la reutilización de código

Las funciones en Python son bloques de código organizados y reutilizables que están diseñados para realizar una tarea específica. Ofrecen una manera eficiente de dividir un programa en módulos, lo que facilita tanto la lectura como el mantenimiento del código.

Definición de una función

Para definir una función en Python, se utiliza la palabra clave def, seguida del nombre de la función y paréntesis que pueden contener argumentos.

Los argumentos son valores que se pasan a la función para que los utilice en su ejecución. Después de los paréntesis, se coloca un dos puntos (:) y el bloque de código indentado que forma el cuerpo de la función.

```
def saludar(nombre):
    print(f"Hola, {nombre}!")
```

Para ejecutar una función, se escribe su nombre seguido de paréntesis. Si la función espera argumentos, se deben proporcionar dentro de los paréntesis.

```
saludar("Mundo")
```

Valores de retorno

Las funciones pueden devolver valores utilizando la palabra clave return. Esto permite que el resultado de una función se asigne a una variable o se utilice de cualquier otra manera.

```
def sumar(a, b):
    return a + b
resultado = sumar(5, 3)
print(resultado)  # Imprime: 8
```

Argumentos predeterminados

Python permite definir valores predeterminados para los argumentos de las funciones. Esto significa que la función puede ser llamada con menos argumentos de los que espera, utilizando los valores predeterminados para los faltantes.

```
def imprimir_mensaje(mensaje="Hola, Python!"):
  print(mensaje)


imprimir_mensaje()  # Utiliza el valor predeterminado
imprimir_mensaje("Otro mensaje")  # Sobrescribe el valor predeterminado
```

Argumentos de palabra clave

Al llamar a una función, puedes especificar argumentos por nombre, lo que te permite ignorar el orden en el que se definen los parámetros y hacer el código más claro.

```
def describir_persona(nombre, edad):
  print(f"Nombre: {nombre}, Edad: {edad}")


describir_persona(edad=30, nombre="Ana")
```

args y *kwargs

Python ofrece una sintaxis especial para pasar un número variable de argumentos a una función: *args para listas de argumentos no nombrados y **kwargs para argumentos nombrados.

```
def funcion_con_varios_argumentos(*args, **kwargs):
        print(args)   # Tupla de argumentos no nombrados
        print(kwargs)  # Diccionario de argumentos nombrados


funcion_con_varios_argumentos(1, 2, tres=3, cuatro=4)
```

Esto permite poder coger de forma genérica cualquier número de argumentos en orden o argumentos con nombre, pudiendo crear funciones que respondan de forma dinámica a lo que introduzcamos en ella.