

Sintaxis de Python

Edison Achalma

Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga

Resumen

Este abstract será actualizado una vez que se complete el contenido final del artículo.

Tabla de contenidos

Introduction	2
0.1 Motivación	2
0.2 Intuición	2
0.3 Modelo mental	2
0.4 Desarrollo	2
0.4.1 1. Indentación: la regla de oro	2
0.4.2 2. Variables: creación sin declaración	2
0.4.3 3. Comentarios: documenta tu código	3
0.4.4 4. Convenciones de sintaxis (PEP 8)	3
0.5 Ejemplos	3
0.6 Errores comunes	4
0.7 Buenas prácticas	4
0.8 Ejercicio	4
0.9 Síntesis	5
0.10 Qué sigue	5
1 Publicaciones Similares	5

Edison Achalma  <https://orcid.org/0000-0001-6996-3364>

El autor no tiene conflictos de interés que revelar. Los roles de autor se clasificaron utilizando la taxonomía de roles de colaborador (CRediT; <https://credit.niso.org/>) de la siguiente manera: Edison Achalma: conceptualización, metodología, análisis formal, investigación, recursos, curación de datos, redacción, visualización, supervisión, administración del proyecto

La correspondencia relativa a este artículo debe dirigirse a Edison Achalma, Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga, Portal Independencia N° 57, Ayacucho, AYA 5001, Perú, Email: elmer.achalma.09@unsch.edu.pe

Sintaxis de Python

0.1 Motivación

La sintaxis es el conjunto de reglas que definen cómo se escribe código válido en Python. A diferencia de otros lenguajes donde la sintaxis es más permisiva, Python fue diseñado para ser **legible y consistente**. Entender la sintaxis desde el principio te ahorrará errores frustrantes y hará que tu código sea más claro para otros (y para ti mismo cuando lo releas).

0.2 Intuición

Piensa en la sintaxis de Python como las reglas gramaticales de un idioma. En español, “El perro come” es válido; “Perro el come” suena mal. En Python, la indentación es como la puntuación: no es opcional, es parte de la gramática. Si olvidas los espacios, Python no entiende dónde termina un bloque y empieza otro, igual que si escribieras una oración sin puntos.

0.3 Modelo mental

La sintaxis de Python se basa en tres pilares fundamentales:

1. **Indentación como estructura:** en lugar de llaves ({}) o palabras clave (begin/end), Python usa espacios para agrupar código. Esto obliga a escribir código limpio y legible.
2. **Instrucciones línea por línea:** cada línea suele ser una instrucción completa. No necesitas punto y coma (;) al final.
3. **Comentarios documentan, no ejecutan:** todo lo que escribas después de # es ignorado por Python. Son para ti y para otros humanos.

0.4 Desarrollo

0.4.1 1. Indentación: la regla de oro

En Python, la indentación **define bloques de código**. Esto aplica a:

- Condicionales (if, elif, else)
- Bucles (for, while)
- Funciones (def)
- Clases (class)

```
if 5 > 2:
    print("Cinco es mayor que dos") # Esta línea está indentada
    print("Esto también pertenece al if")
print("Esto ya está fuera del if") # Sin indentación = fuera del bloque
```

Reglas de indentación:

- Debe haber **al menos un espacio** o un tabulador.
- El **número de espacios debe ser consistente** dentro del mismo bloque.
- La convención más común es **4 espacios** (recomendado por PEP 8).

0.4.2 2. Variables: creación sin declaración

En Python, **no declaras** variables. Se crean cuando les asignas un valor:

```
nombre = "Ana"      # Variable de texto (str)
edad = 25           # Variable numérica (int)
precio = 99.99      # Variable decimal (float)
es_valido = True    # Variable booleana (bool)
```

Python infiere el tipo automáticamente. Las variables pueden cambiar de tipo durante la ejecución (tipado dinámico).

0.4.3 3. Comentarios: documenta tu código

Los comentarios son líneas que Python ignora completamente. Sirven para explicar qué hace tu código:

```
# Esto es un comentario de una línea
print("Hola") # También puedes poner comentarios al final de una línea

# Los comentarios son útiles para:
# - Explicar lógica compleja
# - Desactivar temporalmente código
# - Dejar notas para ti mismo o para otros
```

0.4.4 4. Convenciones de sintaxis (PEP 8)

- **Sensibilidad a mayúsculas:** miVariable y mivariable son diferentes.
- **Nombres en minúsculas con guiones bajos:** nombre_usuario (no nombreUsuario).
- **Constantes en mayúsculas:** MAXIMO = 100.
- **Una línea en blanco** entre funciones y clases para legibilidad.

0.5 Ejemplos

Mínimo — Un programa que combina sintaxis válida:

```
# Calcular descuento simple
precio = 100
descuento = 20
precio_final = precio - descuento
print("Precio final:", precio_final)
# Salida: Precio final: 80
```

Progresivo — Indentación y bloques correctos vs incorrectos:

```
# CORRECTO - mismo número de espacios (4)
if 10 > 5:
    print("Diez es mayor")
    print("Este mensaje también está dentro del if")

# INCORRECTO - indentación inconsistente
if 10 > 5:
    print("Diez es mayor")
    print("Este mensaje tiene 2 espacios") # Error: indentación inconsistente
```

Real — Un script que usa sintaxis básica con comentarios:

```
# config.py - Configuración simple de usuario
nombre = input("¿Cómo te llamas? ") # Captura entrada del usuario
print(f"Hola, {nombre}")             # f-string: formatea cadenas

# Ejecución condicional
if nombre == "admin":
    print("Modo administrador activado")
    print("Tienes acceso completo")
else:
    print("Modo usuario")
    print("Acceso limitado")
```

0.6 Errores comunes

Error	Síntoma	Cómo detectarlo
Indentación faltante	IndentationError: expected an indented block	Busca la línea marcada; debe tener espacios antes del código
Indentación inconsistente	IndentationError: unindent does not match any outer indentation level	Usa espacios o tabs, pero NO mezcles ambos
Confundir = con ==	La variable se reasigna en lugar de comparar	Revisa condicionales: = es asignación, == es comparación
Olvidar cerrar paréntesis/comillas	SyntaxError: unexpected EOF while parsing	Cuenta paréntesis y comillas; deben abrirse y cerrarse
Espacios en nombres de variables	SyntaxError: invalid syntax	Las variables no pueden tener espacios: mi nombre → mi_nombre

0.7 Buenas prácticas

- **Usa 4 espacios para indentar** (configura tu editor para que TAB inserte 4 espacios).
- **Comenta el “por qué”, no el “qué”**: el código ya dice qué hace.
- **Nombra variables de forma descriptiva**: precio_con_descuento en lugar de pcd.
- **No uses palabras reservadas** como variables (print, if, while, class).
- **Agrupar lógica relacionada** con líneas en blanco para separar secciones.

0.8 Ejercicio

Básico — Corrección de errores

El siguiente código tiene **4 errores de sintaxis**. Encuéntralos y corrígelos:

```
# calculadora simple
numero1 = 10
numero2 = 5
suma = numero1 + numero2
```

```
print("La suma es", suma)
if suma > 10:
print("La suma es mayor que 10")
    resta = numero1 - numero2
print("La resta es", resta)
print("Fin del programa")
```

Pista: busca problemas de indentación y nombres mal escritos.

Solución (no mires hasta haberlo intentado):

```
# calculadora simple
numero1 = 10
numero2 = 5
suma = numero1 + numero2
print("La suma es", suma)
if suma > 10:
    print("La suma es mayor que 10") # Faltaba indentación
    resta = numero1 - numero2
print("La resta es", resta)
print("Fin del programa") # La indentación extra no es necesaria
```

0.9 Síntesis

- **Indentación = bloques de código:** esencial, no opcional (4 espacios por convención).
- **Variables se crean al asignar:** no necesitan declaración previa.
- **Comentarios con #:** documentan, no ejecutan.
- **Sintaxis sensible a mayúsculas:** Variable ≠ variable.
- **Usa PEP 8:** guía de estilo para código limpio y consistente.

0.10 Qué sigue

- **Relacionado:** variables y tipos de datos (manejo de texto, números, booleanos)
- **Siguiente tema:** Variables y tipos de datos — aprenderás a almacenar y manipular información con distintos tipos de datos en Python.

1 Publicaciones Similares

Si te interesó este artículo, te recomendamos que explores otros blogs y recursos relacionados que pueden ampliar tus conocimientos. Aquí te dejo algunas sugerencias:

1. [!\[\]\(4695f05050b0d393767d0512587d4e50_img.jpg\) Instalacion De Anaconda](#)
2. [!\[\]\(e6380cce6342e403c00cb7c9feb7e762_img.jpg\) Configurar Entorno Virtual Python Anaconda](#)
3. [!\[\]\(7528551b00a221bf92619f04e9b6fdc4_img.jpg\) 01 Introducion A La Programacion Con Python](#)
4. [!\[\]\(61103e278339dd729c6548dc05b28450_img.jpg\) 02 Variables Expresiones Y Statements Con Python](#)
5. [!\[\]\(c54f4cb3d90acdac1abe98c1a1e231a3_img.jpg\) 03 Objetos De Python](#)
6. [!\[\]\(3444f54eb1f9043b7145ad63f82ea286_img.jpg\) 04 Ejecucion Condicional Con Python](#)
7. [!\[\]\(dde65152fd8669784bc57e6c4616744a_img.jpg\) 05 Iteraciones Con Python](#)

8.  06 Funciones Con Python
9.  07 Dataframes Con Python
10.  08 Prediccion Y Metrica De Performance Con Python
11.  09 Metodos De Machine Learning Para Clasificacion Con Python
12.  10 Metodos De Machine Learning Para Regresion Con Python
13.  Visualizacion De Datos Con Python

Esperamos que encuentres estas publicaciones igualmente interesantes y útiles. ¡Disfruta de la lectura!