Documentación Completa de Sintaxis de Python

Índice

Sintaxis Básica de Python

Variables y Tipos de Datos

```python
# Enteros
x = 10

# Flotantes
y = 3.14

# Cadenas
nombre = "Python"
```

```
# Booleanos
es_verdadero = True
es_falso = False

# Listas
lista = [1, 2, 3, "a", "b"]

# Tuplas (inmutables)
tupla = (1, 2, 3)

# Diccionarios
diccionario = {"clave": "valor", "edad": 25}

# Conjuntos
conjunto = {1, 2, 3}
```

Operadores

```python
# Aritméticos
suma = 5 + 3
resta = 5 - 3
multiplicacion = 5 * 3
division = 5 / 3
division_entera = 5 // 3
modulo = 5 % 3
potencia = 5 ** 3

# Comparación
```

```
igual = 5 == 3
diferente = 5 != 3
mayor = 5 > 3
menor = 5 < 3
mayor_igual = 5 >= 3
menor_igual = 5 <= 3

# Lógicos
y_logico = True and False
o_logico = True or False
negacion = not True
```

Estructuras de Control

Condicionales

```python
# if-elif-else
if x > 10:
    print("Mayor que 10")
elif x == 10:
    print("Igual a 10")
else:
    print("Menor que 10")
```

Bucles

```python
# for
for i in range(5):
    print(i)

for elemento in lista:
    print(elemento)

# while
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Control de Bucles
```python
# break
for i in range(10):
    if i == 5:
        break
    print(i)

# continue
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Funciones

Definición y llamada

```python
def saludar(nombre):
    return f"Hola, {nombre}"

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

Parámetros

```python
# Parámetros por defecto
def potencia(base, exponente=2):
    return base ** exponente

# Argumentos nombrados
resultado = potencia(exponente=3, base=2)

# Argumentos variables (*args y **kwargs)
def funcion_var_args(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)

funcion_var_args(1, 2, 3, nombre="Juan", edad=25)
```

```

### Funciones Lambda

```python
cuadrado = lambda x: x ** 2
print(cuadrado(5))  # 25
```

Clases y POO

Clases básicas

```python
class Persona:
    # Atributo de clase
    especie = "Humano"

    # Constructor
    def __init__(self, nombre, edad):
        # Atributos de instancia
        self.nombre = nombre
        self.edad = edad

    # Método de instancia
    def presentarse(self):
        return f"Soy {self.nombre} y tengo {self.edad} años"

    # Método de clase
```
```

```
    @classmethod
    def desde_nacimiento(cls, nombre, año_nacimiento):
        año_actual = 2023
        edad = año_actual - año_nacimiento
        return cls(nombre, edad)

    # Método estático
    @staticmethod
    def es_mayor_edad(edad):
        return edad >= 18

# Uso
persona1 = Persona("Ana", 25)
print(persona1.presentarse())

persona2 = Persona.desde_nacimiento("Carlos", 1990)
print(persona2.presentarse())
```

Herencia

```python
class Estudiante(Persona):
    def __init__(self, nombre, edad, carrera):
        super().__init__(nombre, edad)
        self.carrera = carrera

    def presentarse(self):
        return f"{super().presentarse()} y estudio {self.carrera}"
```

```
estudiante = Estudiante("Luisa", 20, "Informática")
print(estudiante.presentarse())
```

Manejo de Excepciones

```python
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("No se puede dividir por cero")
except Exception as e:
    print(f"Ocurrió un error: {e}")
else:
    print("Todo salió bien")
finally:
    print("Esto se ejecuta siempre")
```

Módulos y Paquetes

Importación

```python
# Importar módulo completo
import math
```

```
print(math.sqrt(16))

# Importar con alias
import numpy as np

# Importar elementos específicos
from collections import Counter

# Importar todo (no recomendado)
from math import *
```

Creación de módulos
```
mi_paquete/
  __init__.py
  modulo1.py
  modulo2.py
```

Pandas

Estructuras de datos
```python
import pandas as pd

# Series (1D)
```

```
s = pd.Series([1, 3, 5, 7], index=['a', 'b', 'c', 'd'])

# DataFrame (2D)
data = {
   'Nombre': ['Juan', 'Ana', 'Carlos'],
   'Edad': [25, 30, 35],
   'Ciudad': ['Madrid', 'Barcelona', 'Valencia']
}
df = pd.DataFrame(data)
```

Operaciones comunes

```python
# Leer datos
df = pd.read_csv('datos.csv')
df = pd.read_excel('datos.xlsx')

# Exploración
df.head()      # Primeras filas
df.info()      # Información del DataFrame
df.describe()  # Estadísticas descriptivas

# Selección
df['Nombre']       # Columna
df.loc[0]          # Fila por índice
df.iloc[0:2]       # Filas por posición

# Filtrado
df[df['Edad'] > 30]                  # Filtro simple
```

```
df.query('Edad > 30 and Ciudad == "Madrid"')  # Filtro con query

# Manipulación
df['Nueva_Col'] = df['Edad'] * 2       # Nueva columna
df.drop('Ciudad', axis=1, inplace=True) # Eliminar columna

# Agrupación
df.groupby('Ciudad')['Edad'].mean()     # Media de edad por ciudad

# Valores nulos
df.isnull()        # Detectar nulos
df.dropna()        # Eliminar filas con nulos
df.fillna(0)       # Rellenar nulos con 0
```

SQLite3

Conexión y operaciones básicas

```python
import sqlite3

# Conexión a la base de datos
conexion = sqlite3.connect('mi_db.db')
cursor = conexion.cursor()

# Crear tabla
cursor.execute('''
```

```
    CREATE TABLE IF NOT EXISTS usuarios (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nombre TEXT NOT NULL,
        edad INTEGER
    )
''')

# Insertar datos
cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (?, ?)", ('Juan', 25))
conexion.commit()

# Consultar datos
cursor.execute("SELECT * FROM usuarios")
filas = cursor.fetchall()
for fila in filas:
    print(fila)

# Actualizar datos
cursor.execute("UPDATE usuarios SET edad = ? WHERE nombre = ?", (26, 'Juan'))
conexion.commit()

# Eliminar datos
cursor.execute("DELETE FROM usuarios WHERE id = ?", (1,))
conexion.commit()

# Cerrar conexión
conexion.close()
```

Uso con pandas

```python
import pandas as pd
import sqlite3

conexion = sqlite3.connect('mi_db.db')

# Leer datos de SQL a DataFrame
df = pd.read_sql("SELECT * FROM usuarios", conexion)

# Escribir DataFrame a SQL
df.to_sql('usuarios', conexion, if_exists='replace', index=False)

conexion.close()
```

Sockets

Socket TCP básico

Servidor:

```python
import socket

# Crear socket TCP/IP
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
# Vincular socket a puerto
servidor.bind(('localhost', 65432))

# Escuchar conexiones
servidor.listen(1)
print("Esperando conexión...")

# Aceptar conexión
conexion, direccion = servidor.accept()
print(f"Conexión establecida desde {direccion}")

try:
    while True:
        # Recibir datos
        datos = conexion.recv(1024)
        if not datos:
            break
        print(f"Recibido: {datos.decode()}")

        # Enviar respuesta
        conexion.sendall(datos.upper())
finally:
    # Cerrar conexión
    conexion.close()
    servidor.close()
```

Cliente:

```python
```

```
import socket

# Crear socket TCP/IP
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conectar al servidor
cliente.connect(('localhost', 65432))

try:
    # Enviar datos
    mensaje = "Hola, servidor!"
    cliente.sendall(mensaje.encode())

    # Recibir respuesta
    datos = cliente.recv(1024)
    print(f"Recibido: {datos.decode()}")
finally:
    cliente.close()
```

Socket UDP

Servidor:

```python
import socket

servidor = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
servidor.bind(('localhost', 65432))
```

```
print("Esperando mensajes...")
while True:
    datos, direccion = servidor.recvfrom(1024)
    print(f"Recibido de {direccion}: {datos.decode()}")
    servidor.sendto(datos.upper(), direccion)
```

Cliente:

```python
import socket

cliente = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

mensaje = "Hola, servidor UDP!"
cliente.sendto(mensaje.encode(), ('localhost', 65432))

datos, servidor = cliente.recvfrom(1024)
print(f"Recibido: {datos.decode()}")
cliente.close()
```

Tkinter

Ventana básica

```python
import tkinter as tk
from tkinter import messagebox
```

```
# Crear ventana principal
ventana = tk.Tk()
ventana.title("Mi Aplicación")
ventana.geometry("400x300")

# Función para el botón
def saludar():
    nombre = entrada.get()
    messagebox.showinfo("Saludo", f"Hola, {nombre}!")

# Widgets
etiqueta = tk.Label(ventana, text="Ingresa tu nombre:")
etiqueta.pack(pady=10)

entrada = tk.Entry(ventana, width=30)
entrada.pack(pady=10)

boton = tk.Button(ventana, text="Saludar", command=saludar)
boton.pack(pady=10)

# Bucle principal
ventana.mainloop()
```

Widgets comunes

```python
# Frame
marco = tk.Frame(ventana, padx=10, pady=10)
```

```
marco.pack()

# Checkbutton
check_var = tk.BooleanVar()
check = tk.Checkbutton(marco, text="Acepto términos", variable=check_var)
check.pack()

# Radiobutton
radio_var = tk.StringVar(value="opcion1")
radio1 = tk.Radiobutton(marco, text="Opción 1", variable=radio_var, value="opcion1")
radio2 = tk.Radiobutton(marco, text="Opción 2", variable=radio_var, value="opcion2")
radio1.pack()
radio2.pack()

# Listbox
lista = tk.Listbox(marco)
lista.insert(1, "Elemento 1")
lista.insert(2, "Elemento 2")
lista.pack()

# Combobox (necesario ttk)
from tkinter import ttk
combo = ttk.Combobox(marco, values=["Opción 1", "Opción 2", "Opción 3"])
combo.pack()

# Text
area_texto = tk.Text(marco, width=40, height=10)
area_texto.pack()
```

```
# Scrollbar
scroll = tk.Scrollbar(marco, command=area_texto.yview)
scroll.pack(side=tk.RIGHT, fill=tk.Y)
area_texto.config(yscrollcommand=scroll.set)

# Menú
barra_menu = tk.Menu(ventana)
menu_archivo = tk.Menu(barra_menu, tearoff=0)
menu_archivo.add_command(label="Abrir")
menu_archivo.add_separator()
menu_archivo.add_command(label="Salir", command=ventana.quit)
barra_menu.add_cascade(label="Archivo", menu=menu_archivo)
ventana.config(menu=barra_menu)
```

Diseño con grid y place

```python
# Grid
etiqueta.grid(row=0, column=0, sticky="w")
entrada.grid(row=0, column=1)
boton.grid(row=1, columnspan=2)

# Place
boton.place(x=100, y=50)
```

Esta documentación cubre los aspectos esenciales de Python y las librerías mencionadas. Cada sección podría expandirse con más detalles y ejemplos específicos según las necesidades.