📖 Manual de Arquitectura y Lógica en Python

Documentación de Referencia para el Equipo de Desarrollo Responsable: Ander

---

1. Estructuras de Control: Los Condicionales (El Corazón de la Lógica)

En programación, un **condicional** es una instrucción o grupo de instrucciones que se pueden ejecutar o no en función del valor de una condición booleana. Es, en esencia, la implementación del pensamiento lógico humano en el software.

¿Por qué son vitales?

Sin condicionales, los programas serían "scripts" lineales incapaces de reaccionar a cambios en el entorno o a la interacción del usuario. Se utilizan para la toma de decisiones, validación de datos, control de flujo y gestión de errores.

Componentes de la lógica condicional en Python

1. **if**: La entrada principal. Evalúa si una expresión es True.
2. **elif (Else If)**: Permite evaluar múltiples condiciones en cascada de forma eficiente. Solo se ejecuta si las anteriores fallaron.
3. **else**: La cláusula de cierre o "fallback". Se ejecuta si absolutamente ninguna condición previa fue satisfecha.

Operadores Lógicos y de Comparación

Para construir condiciones complejas, nos apoyamos en:

- **Comparación**: == (igualdad), != (desigualdad), >, <, >=, <=.
- **Lógica**: and (y), or (o), not (no/negación).
- **Identidad/Pertenencia**: is (es el mismo objeto), in (está contenido en).

Ejemplo Práctico: Sistema de Gestión de Créditos

Python

```python
# Ejemplo de evaluación multicriterio
ingresos_mensuales = 3500
tiene_deudas = False
puntuacion_crediticia = 750

if ingresos_mensuales > 4000 or (puntuacion_crediticia > 700 and not tiene_deudas):
    print("Crédito aprobado: Nivel Premium")
elif 2000 <= ingresos_mensuales <= 4000 and puntuacion_crediticia > 600:
    print("Crédito aprobado: Nivel Estándar")
else:
    print("Solicitud rechazada: Riesgo elevado")
```

---

2. Bucles e Iteración: La Eficiencia de la Repetición

Un bucle es una estructura que repite un bloque de código mientras se cumpla una condición específica. En el desarrollo moderno, los bucles son cruciales para el procesamiento de grandes conjuntos de datos (Big Data) y la automatización de procesos.

Los dos pilares en Python: for y while

A. El Bucle for (Iteración Definida)

En Python, el for es extremadamente potente porque es un "iterador de objetos". No solo cuenta números; recorre elementos de cualquier objeto iterable (listas, diccionarios, sets, archivos).

- **Uso de range()**: Ideal para repetir una acción un número exacto de veces.
- **Uso de enumerate()**: Fundamental cuando necesitamos el índice y el valor simultáneamente.

B. El Bucle while (Iteración Indefinida)

Se basa exclusivamente en una condición booleana. Es el "vigilante" que mantiene un proceso vivo hasta que sucede algo específico.

- **Precaución**: Es vital asegurar que la condición deje de cumplirse en algún momento para evitar el **bucle infinito**, que podría colapsar la memoria del sistema.

Control Avanzado de Bucles

- **break**: Rompe el bucle de inmediato (útil en búsquedas).
- **continue**: Salta la iteración actual y pasa a la siguiente (útil para omitir datos inválidos).
- **else en bucles**: Una característica única de Python que se ejecuta solo si el bucle terminó normalmente (sin ser interrumpido por un break).

---

3. Listas por Comprensión: El Estilo "Pythonista"

La **comprensión de listas** es un paradigma que permite la creación de nuevas listas basándose en listas existentes, aplicando una expresión y una condición de forma declarativa.

Beneficios

1. **Concisión**: Lo que antes requería 5 líneas de código, ahora se resuelve en una.
2. **Velocidad**: Al estar implementadas en C dentro del intérprete de Python, son notablemente más rápidas que los bucles for tradicionales con .append().

Sintaxis y Ejemplo de Transformación de Datos

[expresión for elemento in iterable if condición]

Python

```
# Queremos obtener el nombre en mayúsculas de los usuarios que son administradores
usuarios = [("ander", "admin"), ("marta", "user"), ("iker", "admin")]

# Tradicional
admins = []
for nombre, rol in usuarios:
    if rol == "admin":
        admins.append(nombre.upper())

# Comprensión de listas (Recomendado)
admins_pro = [nombre.upper() for nombre, rol in usuarios if rol == "admin"]
```

---

4. Argumentos: La Comunicación con las Funciones

Los argumentos son los valores que "inyectamos" en una función para que esta pueda operar. Entender su jerarquía es la diferencia entre un código rígido y uno modular y escalable.

Taxonomía de los Argumentos

- **Argumentos Posicionales**: Los más comunes. Se asignan por su orden de aparición.
- **Argumentos por Nombre (Keyword Arguments)**: Permiten llamar a la función indicando

el nombre del parámetro, lo que aumenta la claridad del código.
- **Parámetros por Defecto**: Permiten que la función tenga valores preestablecidos si el usuario no proporciona unos nuevos.
- **Argumentos Arbitrarios (*args y **kwargs)**:
  - *args: Permite pasar un número variable de argumentos posicionales (se reciben como una tupla).
  - **kwargs: Permite pasar un número variable de argumentos con nombre (se reciben como un diccionario).

---

5. Funciones Lambda: Lógica bajo demanda

Las funciones Lambda son funciones **anónimas** y **efímeras**. Se definen sin nombre y están limitadas a una sola expresión.

¿Por qué y cuándo utilizarlas?

Su propósito no es sustituir a las funciones normales (def), sino actuar como complementos rápidos. Se utilizan principalmente en funciones de orden superior (funciones que reciben otras funciones como parámetros).

- **map()**: Aplica una lambda a cada elemento de una lista.
- **filter()**: Filtra elementos basándose en una condición lambda.
- **sorted()**: Define criterios de ordenación complejos.

Ejemplo de uso en ordenación:

Python

```
empleados = [("Luis", 30000), ("Elena", 45000), ("Juan", 25000)]
# Ordenar por salario (el segundo elemento de la tupla)
empleados_ordenados = sorted(empleados, key=lambda emp: emp[1])
```

---

6. Paquetes PIP y Ecosistema de Terceros

PIP (Pip Installs Packages) es el gestor de paquetes oficial de Python. Es el puente que conecta nuestra aplicación con **PyPI**, un repositorio con cientos de miles de librerías listas para usar.

Gestión Profesional de Paquetes

En un entorno de equipo, no basta con instalar paquetes. Debemos asegurar la reproducibilidad del proyecto:

1. **Entornos Virtuales (venv)**: Creamos un ecosistema aislado para cada proyecto. Esto evita que una actualización en el Proyecto A rompa el Proyecto B.
2. **requirements.txt**: Es el manifiesto de nuestro proyecto. Contiene la lista de todas las librerías y sus versiones exactas.
3. **Instalación Masiva**: Con el comando pip install -r requirements.txt, un nuevo desarrollador puede configurar todo su entorno en segundos.