**Anotação: APIs e Operadores em Python**

**🔗 API (Application Programming Interface)**

* **Definição**: Conjunto de regras que permite que programas se comuniquem
* **Analogia com Operadores**:
  + Assim como operadores são "interfaces" para operações básicas em Python, APIs são "interfaces" para serviços externos
  + Exemplo: + é uma "API" para adição, como endpoints são APIs para serviços web

**🔧 Operadores em APIs REST (Comparação)**

1. **CRUD** (Operações básicas) ↔ **Operadores Aritméticos**
   * POST (Create) → + (Adição)
   * GET (Read) → = (Atribuição)
   * PUT/PATCH (Update) → += (Atribuição com operação)
   * DELETE (Delete) → del (Remoção)
2. **Filtros** ↔ **Operadores de Comparação**
   * GET /users?age\_\_gt=18 → > (Maior que)
   * GET /users?active=true → == (Igualdade)
3. **Lógica em Queries** ↔ **Operadores Lógicos**
   * GET /users?active=true&age\_\_gt=18 → and (E lógico)
   * GET /users?q=(name:John OR age:30) → or (OU lógico)

**💻 Exemplo Prático (Flask API com Operadores)**

from flask import Flask, request

app = Flask(\_\_name\_\_)

users = [{"id": 1, "name": "Ana", "age": 25}]

# Operador AND implícito na query

@app.get("/users")

def get\_users():

name\_filter = request.args.get("name") # Operador ==

age\_filter = int(request.args.get("age", 0)) # Operador >

# Uso combinado de operadores:

result = [

user for user in users

if (not name\_filter or user["name"] == name\_filter) # Operador OR implícito

and user["age"] > age\_filter # Operador AND

]

return {"users": result}

**⚙️ Operadores Especiais em APIs**

1. **Módulo (%)**: Útil para paginação

Page = 5 % 3 # Calcula offsets (resto 2)

1. **Ternário**: Simplifica respostas condicionais

success = True

return {"status": "OK" if success else "Error"}

1. **In/Not In**: Filtros de inclusão

if "admin" in user\_roles: # Verifica permissões

grant\_access()

**🛠 Casos Reais**

* **Validação com Operadores**:

if not (18 <= age <= 65): # Operadores de comparação encadeados

return {"error": "Idade inválida"}, 400

* **Atualização Parcial (PATCH)**:

user["age"] += 1 # Operador += para incremento

**🧠 Resumo Visual**

API REST Python Operator

----------- ---------------

Create (POST) + (Adiciona)

Read (GET) = (Atribui)

Update (PUT) += (Atualiza)

Delete (DELETE) del (Remove)

Filtros >, <, == (Comparação)

Lógica de Query and, or, not

Essa relação mostra como operadores são a "API nativa" do Python para operações básicas, enquanto APIs REST estendem esse conceito para serviços distribuídos. Ambos seguem princípios similares de interface clara e operações bem definidas.