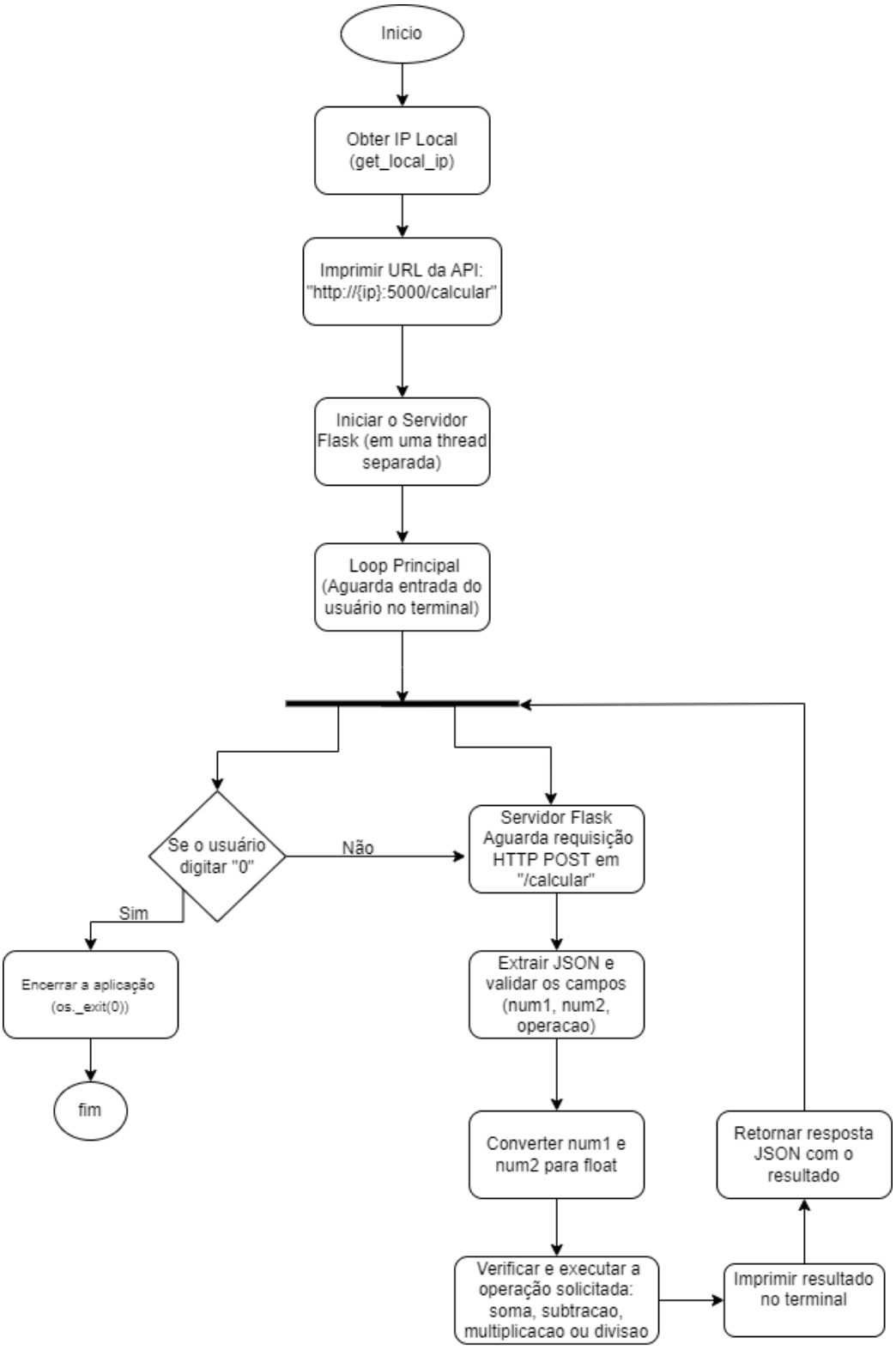
Calculadora HTTP em python

1. Fluxo e comportamento do programa:



2. Funcionalidades chave

I. **Obtenção do IP local e exibição da URL da API:**
O programa utiliza o módulo socket para identificar o IP local da máquina e, em seguida, constrói e imprime no terminal a URL de acesso à API (http://<ip>:5000/calcular).

II. **Execução do servidor Flask em uma thread separada:**
A aplicação Flask, que gerencia a rota /calcular, é iniciada em uma thread distinta. Isso permite que o servidor aceite requisições HTTP sem bloquear o loop principal que aguarda comandos do usuário.

III. **Processamento de requisições e operações matemáticas:**
Ao receber uma requisição POST na rota /calcular, o programa:

a. Extrai e valida os dados enviados no JSON (garantindo que os campos num1, num2 e operacao estejam presentes).
b. Converte os valores num1 e num2 para números (tipo float).
c. Executa a operação matemática solicitada (soma, subtração, multiplicação ou divisão), incluindo a validação para evitar divisão por zero.
d. Retorna o resultado da operação em formato JSON.

IV. **Feedback no terminal:**
Para facilitar o monitoramento, o programa imprime no terminal tanto os resultados das operações quanto eventuais mensagens de erro, permitindo que o usuário veja em tempo real o processamento das requisições.

V. **Controle interativo da execução:**
Um loop principal aguarda a entrada do usuário no terminal. Quando o usuário digita 0, o programa encerra imediatamente todas as threads e finaliza a aplicação utilizando os._exit(0).

3. Imports

1. **Flask**:
    a. **Motivo:** O Flask é um microframework web para Python, utilizado para criar aplicações web e APIs de forma simples e rápida.
    b. **Implementação:** Criamos uma instância do Flask com app = Flask(__name__) para definir a aplicação.
2. **request**:
    a. **Motivo:** O objeto request é usado para acessar os dados enviados pelo cliente na requisição HTTP, como o corpo do JSON enviado para a rota.
    b. **Implementação:** Dentro da função da rota /calcular, usamos data = request.get_json() para extrair os dados da requisição.
3. **jsonify**:
    a. **Motivo:** Facilita a conversão de dicionários Python em respostas JSON formatadas corretamente, garantindo que os cabeçalhos HTTP sejam configurados para JSON.
    b. **Implementação:** Ao final da função, retornamos a resposta com return jsonify({'resultado': resultado}) ou mensagens de erro no mesmo formato.
4. **import socket**

    Motivo: O módulo socket fornece acesso a funções de rede de baixo nível.

    Implementação:

    a. É utilizado na função get_local_ip() para criar um socket temporário que se conecta a um endereço (nesse caso, '10.255.255.255' numa porta arbitrária) para determinar o IP local da máquina.
    b. Se a tentativa de conexão falhar, o IP padrão 127.0.0.1 é usado.
5. **import threading**

    Motivo: Permite a execução de múltiplas tarefas de forma concorrente através de threads.

    Implementação:

    a. Usamos o threading.Thread para executar o servidor Flask em uma thread separada.

b. Isso é necessário porque o servidor Flask é “bloqueante”; ou seja, se executado na thread principal, não permitiria a execução simultânea do loop que aguarda a entrada do usuário para encerrar a aplicação.

6. **import os**

   Motivo: O módulo os possibilita a interação com funcionalidades do sistema operacional.

   Implementação:

   a. No loop principal, quando o usuário digita 0, usamos os._exit(0) para forçar a finalização do processo.

   b. Essa função encerra imediatamente o programa, interrompendo todas as threads ativas.

Resumo

O programa implementa uma API simples usando Flask que realiza operações matemáticas (soma, subtração, multiplicação e divisão) a partir de dados enviados via JSON. Ele obtém e exibe o IP local para acesso, roda o servidor em uma thread separada e permite o encerramento da aplicação via entrada do usuário no terminal.