As variáveis devem ser tratadas. Os erros devem ser previstos.

**Try Except**

O try serve para mandar o programa tentar fazer o determinado trecho do código e caso ele não consiga, fará o except.

Eles servem para o código não quebrar caso haja um erro.

Ex:

def escolher\_opcao():

    try:

        opcao\_escolhida = int(input('Escolha uma opção: '))

        # opcao\_escolhida = int(opcao\_escolhida)

        if opcao\_escolhida == 1:

            print('Cadastrar restaurante')

        elif opcao\_escolhida == 2:

            print('Listar restaurante')

        elif opcao\_escolhida == 3:

            print('Ativar restaurante')

        elif opcao\_escolhida == 4:

            finalizar\_app

        else:

            opcao\_invalida()

    except:

        opcao\_invalida()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Listas**

Para criar uma lista, basta colocar o nome dela com colchetes depois:  
Lista = []

Para colocar algo dentro da lista, usa-se o nome da lista, ponto, append. No append, coloca-se o que será colocado na lista.

Ex:

restaurantes = []

def cadastrar\_novo\_restaurante():

    os.system('cls')

    print('Cadastro de novos restaurantes\n')

*nome\_do\_restaurante* = input('Digite o nome do restaurante que deseja cadastrar: ')

**restaurantes.append**(*nome\_do\_restaurante*)

Obs: por padrão, o input já deixa o tipo como string.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Laço de repetição for**

Ex:

for restaurante in restaurantes:

        print(f'.{restaurante}')

Exatamente! Em Python, a variável restaurante é criada automaticamente no próprio laço for.

Quando você escreve:

for restaurante in restaurantes:

O Python faz o seguinte:

1. Cria a variável restaurante no início do laço.
2. A cada iteração, a variável restaurante assume o valor de um dos itens da lista restaurantes. Ou seja:
   * Na primeira iteração, restaurante será 'Pizza'.
   * Na segunda iteração, restaurante será 'Sushi'.
3. O laço continua até que todos os elementos da lista sejam processados.

Essa criação direta da variável dentro do for é um recurso da linguagem para facilitar o código, sem a necessidade de declarar a variável restaurante antes do laço.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Laço de repetição while**

Imagine que estamos desenvolvendo um programa para uma aplicação de entrada de dados em que precisamos coletar informações do usuário. Uma das informações que precisamos é um número positivo, e queremos garantir que o usuário forneça um valor válido.

Nosso desafio é criar um código que solicite ao usuário que insira um número positivo. No entanto, como não podemos confiar no usuário para fornecer um valor válido na primeira tentativa, precisamos criar uma lógica que permita solicitar novamente até que o usuário finalmente insira um número válido.

Como aprendemos, podemos utilizar a estrutura de repetição for para solicitar esse valor até que essa condição seja satisfeita, como mostrado no código a seguir:

numero = -1

**for** \_ **in** range(3): # Supondo um número máximo de tentativas (3) arbitrário

numero = int(input("Digite um número positivo: "))

**if** numero > 0:

**break**

print("Você digitou:", numero)

Perceba que, para criar esse loop, precisamos definir um número arbitrário de tentativas para que o usuário inserisse esse valor. Isso acontece porque o loop for é utilizado quando se conhece previamente o número de iterações que devem ser realizadas.

Contudo, nós queremos que o usuário digite diversas vezes **até** que ele coloque um valor positivo e não tenha nenhuma limitação.

Neste caso, a estrutura for não consegue satisfazer o que desejamos para nosso código. E agora? O Python tem outra estrutura que podemos utilizar? A resposta é sim!

Essa linguagem oferece duas estruturas de controle de fluxo fundamentais para a execução de blocos de repetição: o **for** e o **while**. Ambas são utilizadas para a implementação de loops, permitindo que um bloco de código seja executado repetidamente enquanto determinadas condições são atendidas.

Como vimos, o loop for é utilizado quando se conhece previamente o número de iterações que devem ser realizadas. Ele é especialmente eficaz ao percorrer elementos em sequências, como listas, tuplas, strings ou ranges.

O loop while, diferente do for, é utilizado quando o número de iterações não é conhecido de antemão, mas ainda assim depende de uma condição específica para manter o bloco de código em repetição. Ele continua a executar o bloco de código enquanto a condição fornecida for avaliada como verdadeira.

A sintaxe do loop while é a seguinte:

**while** condição:

# Bloco de código a ser repetido

O bloco de código dentro do while continuará a ser executado até que a condição se torne falsa, ou seja, ele apenas será executado quando a condição assumir o valor booleano de true. Isso significa que é essencial e necessário garantir que a condição eventualmente se torne falsa para evitar um loop infinito em seu projeto.

Então podemos adaptar nosso projeto anterior para utilizar o while ao invés do for, como vemos no código a seguir:

numero = -1

**while** numero <= 0:

numero = int(input("Digite um número positivo: "))

print("Você digitou:", numero)

Enquanto o número digitado for menor ou igual a zero, o programa continuará pedindo ao usuário que insira um número positivo. Quando o usuário finalmente fornecer um número positivo, o loop while será encerrado e o programa exibirá o número digitado.

Sendo assim, esse é um cenário em que um loop while é mais apropriado, pois não sabemos com antecedência quantas vezes precisaremos solicitar ao usuário que insira um número.

Ou seja, a escolha entre for e while dependerá da natureza específica do problema que você está resolvendo. Entender as nuances e aplicar a estrutura de controle de fluxo mais apropriada contribuirá para um código mais claro e eficiente a depender do seu projeto.