Python: Do Zero a Imersão Dev Agentes de IA

MasterClass Python

Autor: Prof. Guilherme Lima
Linkedin: https://www.linkedin.com/in/guilherme-lima-developer/
GitHub: https://github.com/guilhermeonrails

Índice

Introdução: Sua Aventura Começa Agora

Olá, futuro(a) programador(a)!

Se você abriu este livro, provavelmente está curioso(a) sobre o mundo da programação e, especificamente, sobre Python. Talvez você já tenha ouvido falar que Python é usado em tudo, desde websites até inteligência artificial, e pensou: "Será que eu consigo aprender isso?".

A resposta é um grande e sonoro **SIM**!

Por que Python?

Python foi criado com uma filosofia clara: o código deve ser fácil de ler e de escrever. É uma linguagem incrível, pois sua sintaxe se assemelha muito à língua inglesa, eliminando muitas das complexidades que outras linguagens impõem.

Nosso Objetivo Final

Ao final desta jornada, você não apenas entenderá os blocos fundamentais da programação, mas será capaz de ler e compreender o seguinte trecho de código. Parece grego agora? Perfeito! Em breve, cada linha fará total sentido.

```
from typing import Dict
def perguntar(pergunta: str) -> Dict:
    docs_relacionados = retriever.invoke(pergunta)
    if not docs_relacionados:
        return {"answer": "Não sei."}
    return {"answer": docs_relacionados}

resposta = perguntar("Qual é a política de privacidade?")
print("Resposta ->", resposta["answer"])
```

Os Primeiros Passos no Universo Python

Conversando com o Computador: O que é Programar?

Programar é, essencialmente, dar instruções a um computador em uma linguagem que ele entenda. Pense nisso como escrever uma receita. Cada passo deve ser claro e preciso. Em nossa jornada, a linguagem que usaremos é o Python.

Variáveis: As Caixas de Memória do seu Código

Imagine que você precisa guardar uma informação para usar depois, como o nome de um usuário ou um número. Você usaria uma "caixa" com uma etiqueta, certo? Em programação, essas caixas são chamadas de **variáveis**.

Para criar uma variável em Python, você dá um nome a ela e usa o sinal de igual (`=`) para guardar algo dentro.

```
# Aqui, criamos uma variável chamada 'saudacao' e guardamos o texto "Olá, mundo!" nela.
saudacao = "Olá, mundo!"

# Agora, podemos usar a variável para mostrar o texto na tela.
print(saudacao)
```

> **Resultado:** Olá, mundo!

Tipos de Dados Essenciais: Textos e Verdades

As informações que guardamos nas variáveis têm tipos diferentes. Por enquanto, vamos focar em dois tipos cruciais para o nosso código-alvo.

- **Strings:** Qualquer pedaço de texto em Python é chamado de **string** (ou `str`). Para criar uma string, basta colocar o texto entre aspas duplas (`"`) ou simples (`'`).

```
pergunta_do_usuario = "Qual é a política de privacidade?"
resposta_padrao = "Não sei."
```

- **Números:** Para trabalhar com matemática, usamos números. Os números inteiros (sem casas decimais) são do tipo **int**, e os números com casas decimais são do tipo **float**.

```
# Exemplo de int
idade = 30

# Exemplo de float
preco_combustivel = 5.89
```

- **Booleanos:** Às vezes, só precisamos saber se algo é verdadeiro ou falso. Para isso, usamos o tipo **booleano** (ou `bool`), que só tem dois valores possíveis: `True` (Verdadeiro) e `False` (Falso). Note que eles começam com letra maiúscula.

Nesta seção, você aprendeu o básico: o que é programar, como guardar informações em **variáveis** e conheceu os tipos **string**, **int**, **float** e **booleano**. Já é um grande começo!

Guardando e Organizando Dados

No tópico anterior, aprendemos a guardar informações simples. Agora, vamos ver como organizar conjuntos de dados, algo fundamental para o nosso código-alvo.

Dicionários: A Arte de Etiquetar Informações

E se quiséssemos guardar informações mais estruturadas, com etiquetas? Para isso, usamos **dicionários** (ou `dict`). Um dicionário guarda pares de `chave: valor`. A chave é a etiqueta (uma string) e o valor é a informação que você quer guardar. Usamos chaves `{}` para criar dicionários.

```
# Criando um dicionário para representar a resposta do nosso sistema
resultado_final = {
    "answer": "A política de férias permite 30 dias por ano."
}

print(resultado_final)
```

Resultado: {'answer': 'A política de férias permite 30 dias por ano.'}

Listas: Coleções de Itens

Uma **lista** (ou `list`) é uma coleção ordenada de itens. Para criá-la, usamos colchetes `[]` e separamos os itens por vírgulas. No nosso código-alvo, a busca por documentos relacionados retornará uma lista, que pode estar vazia.

```
# Uma lista de strings
docs_encontrados = ["Artigo 1 da Política", "Página 5 do Manual"]

# Uma lista vazia, para o caso de não encontrar nada
docs_vazios = []
```

Objetos e seus Superpoderes (Métodos)

Em Python, quase tudo (strings, listas, dicionários) é um "objeto". Pense em um objeto como algo que não só guarda dados, mas também tem "poderes" ou "ações" embutidas. Essas ações são chamadas de **métodos**. Para usar um método, você coloca um ponto (.) depois do nome da variável e o nome do método. O nosso código usa o método .invoke() de uma ferramenta externa.

Você agora sabe organizar dados em **dicionários** e **listas** e como usar **métodos** para manipular esses dados. Estamos chegando perto!

Ensinando o Código a Tomar Decisões

Um programa seria muito chato se executasse sempre as mesmas instruções. O poder da programação está em tomar decisões com base nas informações disponíveis.

O if: A Encruzilhada do seu Programa

A principal ferramenta para tomar decisões é a declaração if (que significa "se" em inglês). A estrutura é simples: if condicao:. Se a condicao for True, o código dentro do bloco if (o código com um recuo) é executado.

```
usuario_fez_login = True

if usuario_fez_login:
    print("Bem-vindo de volta!")

print("Fim do programa.")
```

Resultado:

Bem-vindo de volta!

Fim do programa.

Comparações e Lógica: `not`

As condições que o `if` avalia geralmente vêm de operadores lógicos. O operador `not` inverte um valor booleano. `not True` se torna `False`, e `not False` se torna `True`.

Aqui está um truque muito comum e poderoso em Python: coleções vazias (como uma lista `[]`) são consideradas `False` quando avaliadas em um `if`. Coleções com qualquer item dentro são consideradas `True`.

Isso nos permite escrever um código mais limpo:

```
documentos_encontrados = [] # Lista vazia

# Este 'if' pergunta: "A lista de documentos NÃO está vazia?"
# Como a lista está vazia (False), 'not' a inverte para True.
if not documentos_encontrados:
    print("Nenhum documento foi encontrado.")
```

Resultado: Nenhum documento foi encontrado.

Essa é exatamente a lógica usada na linha `if not docs_relacionados:` do nosso código-alvo!

Você agora sabe como controlar o fluxo do seu programa com `if` e `not`. A peça final do quebra-cabeça são as funções.

Criando Blocos de Código Reutilizáveis: As Funções

Imagine que você tem um conjunto de instruções que precisa executar várias vezes. Copiar e colar seria ineficiente e propenso a erros. Para resolver isso, usamos **funções**.

O que são Funções e por que elas são Incríveis?

Uma função é um bloco de código nomeado que realiza uma tarefa específica. Você pode "chamar" a função pelo nome sempre que precisar executar essa tarefa. Isso torna o código mais organizado, legível e reutilizável.

Anatomia de uma Função: `def`, Parâmetros e `return`

Vamos quebrar a estrutura para criar uma função:

- `def`: A palavra-chave que diz ao Python: "Estou definindo uma nova função".
- `nome_da_funcao`: O nome que você dá à sua função.
- `()`: Parênteses que podem conter **parâmetros**. Parâmetros são informações que você "passa para dentro" da função para ela usar.
- `:`: Dois pontos que indicam o início do bloco de código da função.
- `return`: A palavra-chave que diz qual informação a função deve "devolver" como resultado.

```
# Definindo uma função simples
def criar_resposta_padrao():
    resposta = {"answer": "Não sei."}
    return resposta

# Chamando a função e guardando o resultado em uma variável
resultado = criar_resposta_padrao()
print(resultado)
```

Resultado: {'answer': 'Não sei.'}

Dicas de Rota: Usando `from typing import Dict`

No nosso código-alvo, você vê anotações como `from typing import Dict` e `-> Dict`. Elas são chamadas de "dicas de tipo" ou **Type Hints**.

- `from typing import Dict` importa o objeto que representa o tipo `dicionário`.
- `-> Dict` depois dos parênteses da função indica que esperamos que a função retorne um dicionário.

Elas não mudam como o código funciona, mas são extremamente úteis para que outros programadores (e você mesmo no futuro!) entendam rapidamente o que a função espera receber e o que ela devolve.

Com o poder das funções, estamos finalmente prontos para decifrar nosso código!

Desvendando o Código Final

Chegou a hora! Com todo o conhecimento que você adquiriu nas seções anteriores, vamos analisar o código-alvo, linha por linha. Todo o seu esforço culmina neste momento.

```
from typing import Dict

def perguntar(pergunta: str) -> Dict:
    docs_relacionados = retriever.invoke(pergunta)

    if not docs_relacionados:
        return {"answer": "Não sei."}

    return {"answer": docs_relacionados}

resposta = perguntar("Qual é a política de privacidade?")
print("Resposta ->", resposta["answer"])
```

Análise Linha por Linha

- `from typing import Dict`: Esta linha, vista na seção sobre funções, importa o objeto `Dict` para que possamos usar a dica de tipo.
- `def perguntar(pergunta: str) -> Dict:`: Estamos definindo uma **função** chamada `perguntar`. Ela aceita um **parâmetro** chamado `pergunta`, que deve ser uma string, e promete retornar um **dicionário**.
- `docs_relacionados = retriever.invoke(pergunta)`: Aqui, usamos uma ferramenta externa (`retriever`) para buscar documentos. Passamos a nossa `pergunta` para o método `.invoke()`, e o resultado (uma lista de documentos) é guardado na **variável** `docs_relacionados`.
- `if not docs_relacionados:`: Este é o nosso **if**. A condição `not docs_relacionados` checa se a lista de documentos está **vazia**. Se a busca não encontrou nada, esta condição é `True`.
- `return {"answer": "Não sei."}`: Se a condição do `if` for verdadeira, a função para aqui e **retorna** um **dicionário** padrão, informando que a resposta é "Não sei.".
- `return {"answer": docs_relacionados}`: Se o programa chegou até aqui, significa que a busca funcionou. A função **retorna** um **dicionário** final com a chave `"answer"` contendo a lista de documentos encontrada.
- `resposta = perguntar("Qual é a política de privacidade?")`: Esta linha mostra como a função é usada. Chamamos `perguntar()` com um texto, e o **dicionário** que ela retorna é guardado na **variável** `resposta`.
- `print("Resposta ->", resposta["answer"])`: Finalmente, esta linha usa a função `print()` para exibir o resultado. Usamos `resposta["answer"]` para acessar o **valor** associado à **chave** `"answer"` no dicionário que a nossa função nos retornou.

Missão Cumprida!

Parabéns! Você fez isso. Você leu, entendeu e dissecou um pedaço de código Python real. Você viajou dos conceitos mais básicos, como variáveis, até a estrutura de uma função, entendendo cada decisão que o código toma.

A programação é uma jornada de aprendizado contínuo. O que você aprendeu aqui são os blocos de construção para absolutamente tudo em Python. Continue praticando, seja curioso e nunca pare de aprender. O universo da programação está agora aberto para você.