Linguagem de Programação Back-End

Prof. Wellington S. S. Silva

Python: Bibliotecas

Introdução: O que é uma Biblioteca para Linguagem Python?

Biblioteca – coleção de Ferramentas, Funções e módulos

No mundo da programação, uma **biblioteca** (ou *library*, em inglês) é como uma grande coleção de ferramentas, funções e módulos que já foram escritos por outras pessoas. Em vez de começar um projeto do zero e criar cada pequena função que você precisa, você pode simplesmente usar essas ferramentas prontas.

Pense em um projeto de construção. Você poderia fabricar cada martelo, chave de fenda e serra que você precisa. Ou, você poderia ir a uma loja de ferramentas e comprar um conjunto de ferramentas já feito e de alta qualidade. As bibliotecas são essa loja de ferramentas. Elas economizam tempo, evitam que você "reinvente a roda" e garantem que o seu código seja mais eficiente e confiável, já que essas bibliotecas são amplamente testadas.

Para usar uma biblioteca em seu código, você geralmente precisa importá-la com um comando simples, como **import biblioteca_desejada**.

Algumas das Bibliotecas mais utilizadas

1. NumPy (Numerical Python)
2. Pandas
3. Matplotlib
4. Scikit-learn
5. TensorFlow e PyTorch
6. Requests
7. Django e Flask
8. Beautiful Soup
9. Seaborn
10. OpenCV (Open Source Computer Vision Library)

Biblioteca – NumPy (Numerical Python)

- **O que é:** É a biblioteca fundamental para computação numérica e científica em Python.
- **Para que serve:** Ela oferece estruturas de dados e funções eficientes para trabalhar com grandes arrays (vetores e matrizes) e realizar operações matemáticas complexas de forma rápida. É a base para muitas outras bibliotecas de ciência de dados.

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import numpy as np

            # 1. Criando arrays NumPy

        # Cria um array unidimensional (vetor) a partir de uma lista Python
vetor = np.array([1, 2, 3, 4, 5])
print("Vetor original:")
print(vetor)

        # Cria um array bidimensional (matriz)
matriz_a = np.array([[1, 2],
                     [3, 4]])
print("\nMatriz A:")
print(matriz_a)

matriz_b = np.array([[5, 6],
                     [7, 8]])
print("\nMatriz B:")
print(matriz_b)

            # 2. Operação matemática em todos os elementos

        # Eleva cada elemento do vetor ao quadrado.
        # Note que a operação é aplicada a cada item de forma eficiente.
vetor_ao_quadrado = vetor ** 2
print("\nCada elemento do vetor elevado ao quadrado:")
print(vetor_ao_quadrado)

            # 3. Multiplicação de matrizes

        # Realiza a multiplicação de matrizes de forma eficiente usando np.dot()
        # O resultado é uma nova matriz.
matriz_resultado = np.dot(matriz_a, matriz_b)
print("\nResultado da multiplicação de Matriz A por Matriz B:")
print(matriz_resultado)
```

Biblioteca – Pandas

- **O que é:** Uma das bibliotecas mais usadas para análise e manipulação de dados.

- **Para que serve:** O Pandas introduz o conceito de **DataFrame**, que é como uma planilha do Excel ou uma tabela de banco de dados, mas dentro do Python. Ele facilita a limpeza, o filtro, a agregação e a visualização de dados de forma intuitiva.

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import pandas as pd

# 1. Criando um DataFrame a partir de um dicionário
dados = {
    'Nome': ['Alice', 'Bob', 'Carlos', 'Diana', 'Eva'],
    'Idade': [25, 30, 35, 22, 28],
    'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte', 'São Paulo', 'Rio de Janeiro']
}

# Cria o DataFrame
df = pd.DataFrame(dados)

print("--- DataFrame Original ---")
print(df)

# 2. Selecionando uma coluna
print("\n--- Apenas a coluna 'Nome' ---")

# Selecionar uma única coluna retorna uma Series (tipo de dado do Pandas)
nomes = df['Nome']
print(nomes)

# 3. Filtrando dados por uma condição
print("\n--- Pessoas com mais de 25 anos ---")

# Cria uma condição (uma Series de True/False) e a usa para filtrar o DataFrame
df_filtrado = df[df['Idade'] > 25]
print(df_filtrado)

# 4. Realizando um cálculo simples
# Calcula a média da idade de todas as pessoas

media_idade = df['Idade'].mean()
print(f"\n--- Média de Idade ---")
print(f"A média de idade das pessoas é: {media_idade:.2f} anos")
```

Biblioteca – Matplotlib

- **O que é:** Uma das bibliotecas de visualização de dados mais antigas e amplamente usadas.

- **Para que serve:** Permite criar uma grande variedade de gráficos estáticos de alta qualidade, como gráficos de linha, barras, dispersão e histogramas, para ajudar a entender e apresentar dados.

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import matplotlib.pyplot as plt
import numpy as np

        # 1. Preparando os dados

meses = ['Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun']
vendas = np.array([150, 200, 250, 300, 280, 320])


        # 2. Criando o gráfico
plt.figure(figsize=(8, 6))  # Cria uma figura para o gráfico com tamanho definido

    # Plota os dados: meses no eixo X e vendas no eixo Y
plt.plot(meses, vendas, marker='o', linestyle='-', color='b')


        # 3. Adicionando rótulos e título
plt.title('Vendas Mensais no Primeiro Semestre', fontsize=16)
plt.xlabel('Mês', fontsize=12)
plt.ylabel('Total de Vendas', fontsize=12)

    # Adiciona uma grade para facilitar a leitura
plt.grid(True)


        # 4. Exibindo o gráfico
plt.show()
```

Biblioteca – Scikit-learn

- **O que é:** A biblioteca mais popular para aprendizado de máquina (Machine Learning) em Python.
- **Para que serve:** Ela oferece uma vasta gama de algoritmos e ferramentas para tarefas de aprendizado de máquina, como classificação, regressão, *clustering* e redução de dimensionalidade. É conhecida por sua simplicidade e consistência na API.

Para instalar a Biblioteca execute o comando abaixo.

```
pip install scikit-learn
```

A seguir temos um exemplo de código utilizando essa biblioteca.

```
from sklearn.linear_model import LinearRegression

# Dados de exemplo
X = [[1], [2], [3], [4]]   # variável independente
y = [2, 4, 6, 8]           # variável dependente

# Criando e treinando o modelo
modelo = LinearRegression()
modelo.fit(X, y)

# Fazendo previsão
print(modelo.predict([[5]]))  # Resultado esperado: ~10
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

                # 1. Gerando dados de exemplo para regressão linear
# x é a nossa variável independente (por exemplo, "horas de estudo")
# y é a nossa variável dependente (por exemplo, "nota no teste")
np.random.seed(0)  # Para garantir que os dados sejam os mesmos em todas as execuções
x = 2 * np.random.rand(100, 1) # 100 valores aleatórios entre 0 e 2
y = 4 + 3 * x + np.random.randn(100, 1) # A equação é y = 4 + 3x, com algum "ruído"

                # 2. Dividindo os dados em conjuntos de treino e teste
# 80% para treino e 20% para teste
x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, test_size=0.2, random_state=42)

                # 3. Criando e treinando o modelo de Regressão Linear
modelo = LinearRegression()
modelo.fit(x_treino, y_treino)

                # 4. Fazendo previsões com o conjunto de teste
y_previsto = modelo.predict(x_teste)

                # 5. Avaliando o modelo
# Calcula o Erro Quadrático Médio (Mean Squared Error)
mse = mean_squared_error(y_teste, y_previsto)
print(f"Erro Quadrático Médio (MSE): {mse:.2f}")

                # 6. Visualizando os resultados
plt.figure(figsize=(10, 6))
# Plota os pontos de dados de teste (os originais)
plt.scatter(x_teste, y_teste, color='blue', label='Dados de Teste Reais')
# Plota a linha de regressão (a previsão do modelo)
plt.plot(x_teste, y_previsto, color='red', linewidth=3, label='Previsão do Modelo')
plt.title('Regressão Linear Simples')
plt.xlabel('Variável X')
plt.ylabel('Variável Y')
plt.legend()
plt.grid(True)
plt.show()
```

Biblioteca – TensorFlow e PyTorch

- **O que é:** As duas principais bibliotecas para **aprendizado profundo** (*Deep Learning*).

- **Para que serve:** São usadas para construir e treinar redes neurais complexas, que são a base de muitas aplicações de IA, como reconhecimento de imagem, processamento de linguagem natural e muito mais.

Para instalar a Biblioteca execute o comando abaixo.

pip install TensorFlow

pip install Torch

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import tensorflow as tf
from tensorflow import keras

        # 1. Carregar o dataset MNIST (imagens de 28x28 pixels)
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

        # 2. Normalizar os dados (0-255 → 0-1)
x_train = x_train / 255.0
x_test = x_test / 255.0

        # 3. Criar o modelo da rede neural
modelo = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),   # transforma 28x28 em vetor de 784 posições
    keras.layers.Dense(128, activation='relu'),   # camada oculta com 128 neurônios
    keras.layers.Dense(10, activation='softmax')  # saída: 10 classes (0 a 9)
])

        # 4. Compilar o modelo
modelo.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

        # 5. Treinar o modelo
modelo.fit(x_train, y_train, epochs=5)

        # 6. Avaliar no conjunto de teste
teste_loss, teste_acc = modelo.evaluate(x_test, y_test)
print(f"Acurácia no teste: {teste_acc:.2f}")
```

```
import torch
import torch.nn as nn
import torch.optim as optim

        # 1. Dados de entrada e saída (queremos aprender a relação y = 2x + 1)
X = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
y = torch.tensor([[3.0], [5.0], [7.0], [9.0]])

        # 2. Definir o modelo (camada linear simples: y = w*x + b)
modelo = nn.Linear(in_features=1, out_features=1)

        # 3. Função de perda (erro quadrático médio)
criterio = nn.MSELoss()

        # 4. Otimizador (Gradiente Descendente usando SGD)
otimizador = optim.SGD(modelo.parameters(), lr=0.01)

        # 5. Treinamento
for epoca in range(500):
    y_pred = modelo(X)              # Forward: previsão

    perda = criterio(y_pred, y)     # Calcular perda

    otimizador.zero_grad()                  # Zerar gradientes acumulados

    perda.backward()                # Backward: calcular gradientes

    otimizador.step()               # Atualizar pesos

    # Mostrar progresso a cada 50 épocas
    if epoca % 50 == 0:
        print(f"Época {epoca}, Perda: {perda.item():.4f}")

        # 6. Testando previsão
entrada = torch.tensor([[5.0]])
saida = modelo(entrada)
print(f"Previsão para x=5: {saida.item():.2f}")
```

Biblioteca – Requests

- **O que é:** Uma biblioteca para fazer requisições HTTP.

- **Para que serve:** Permite que seu código se comunique com a internet de forma simples. É usada para interagir com APIs e baixar conteúdo de sites, por exemplo.

pip install requests

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import requests

                # 1. Definir a URL da API que queremos acessar
        # A URL aponta para a postagem de ID 10 na API de exemplo
url = 'https://jsonplaceholder.typicode.com/posts/10’


                # 2. Fazer a requisição HTTP GET
        # A função requests.get() envia a requisição e retorna um objeto de resposta
response = requests.get(url)


                # 3. Verificar se a requisição foi bem-sucedida (código de status 200)
if response.status_code == 200:


                 # 4. Acessar os dados da resposta no formato JSON
        # A função response.json() converte a resposta para um dicionário Python
    post = response.json()


                # 5. Imprimir os dados de forma legível
    print("--- Dados da Postagem ---")
    print(f"ID do Usuário: {post['userId']}")
    print(f"ID da Postagem: {post['id']}")
    print(f"Título: {post['title']}")
    print(f"Corpo: {post['body']}")

else:
            # Caso a requisição não seja bem-sucedida, imprime o código de erro
    print(f"Erro ao buscar os dados. Código de status: {response.status_code}")
```

Biblioteca – Django e Flask

- **O que é:** Dois dos *frameworks* mais populares para desenvolvimento web.

- **Para que serve:** Embora sejam mais que simples bibliotecas, eles são coleções de módulos que ajudam a construir aplicações web completas. **Django** é um *framework* "completo" (*full-stack*), enquanto **Flask** é mais leve e flexível, ideal para projetos menores.

pip install Flask

A seguir temos um exemplo de código utilizando essa biblioteca.

```
from flask import Flask

        # 1. Cria a instância do aplicativo Flask
app = Flask(__name__)


        # 2. Define a primeira rota para a URL base ('/')
    # Quando alguém acessa a URL raiz, esta função é executada
@app.route('/')
def hello_world():
    return '<h1>Olá, Mundo Maravilhoso! 🌍</h1><p>Bem-vindo à sua primeira página com Flask (Nice!!!) 😉.</p>'


        # 3. Define uma segunda rota que aceita um parâmetro na URL
    # O <nome> na URL será passado como argumento para a função
@app.route('/ola/<nome>')
def ola_nome(nome):
    return f'<h1>Olá, {nome}! 👍😎</h1>'


        # 4. Executa o servidor de desenvolvimento
    # O código dentro deste bloco só roda quando você executa o arquivo diretamente


if __name__ == '__main__':
    app.run(debug=True)
```

Rodando a aplicação feita emFlask

Primeiro, pelo terminal, rode o programa que acabou de criar.

Também pode-se ir na pasta que está o programa criado e rodar a aplicação da maneira abaixo.

Python nomedoprograma.py

Quando o programa estiver rodando, abra um navegador de internet e coloque as URLs abaixo.

http://127.0.0.1:5000/

http://127.0.0.1:5000/ola/Fulano

Biblioteca – Random

- **O que é:** A biblioteca **random** em Python é uma ferramenta essencial para gerar números e escolher elementos de forma aleatória. Ela faz parte da biblioteca padrão do Python, o que significa que você não precisa instalá-la, basta importá-la para usar suas funções.

O nome "**random**" em programação na verdade se refere a "***pseudo-aleatório***". Isso significa que os números gerados não são verdadeiramente aleatórios, mas são criados por um algoritmo matemático que produz uma sequência de números que parece aleatória. Isso é útil porque a sequência pode ser reproduzida se você usar a mesma semente, o que é importante para testes e simulações.

A seguir temos um exemplo de código utilizando essa biblioteca.

Biblioteca – Random

Principais Funções da Biblioteca Random.

A biblioteca random oferece várias funções para diferentes tipos de sorteio. As mais comuns incluem:

random.random(): Retorna um número de ponto flutuante (float) entre 0.0 e 1.0.

random.randint(a, b): Retorna um número inteiro aleatório entre a e b, incluindo ambos os valores. É ideal para simular um dado, por exemplo.

random.choice(sequência): Retorna um elemento aleatório de uma sequência não vazia, como uma lista, uma tupla ou uma string. É a função que usamos para sortear um nome de uma lista.

random.shuffle(lista): Embaralha os itens de uma lista no lugar, ou seja, a lista original é modificada.

random.sample(população, k): Retorna uma nova lista com k elementos únicos escolhidos aleatoriamente da população (que pode ser uma lista ou tupla). É útil para sortear vários vencedores sem repetição.

A seguir temos um exemplo de código utilizando essa biblioteca.

```
import random

        # 1. Definir a lista de nomes que você quer usar
nomes_lista = ["Alice", "Bob", "Carlos", "Diana", "Eva", "Fábio", "Giovana"]

        # 2. Verificar se a lista não está vazia para evitar erros
if nomes_lista:

        # 3. Usar a função random.choice() para escolher um nome aleatoriamente
    nome_escolhido = random.choice(nomes_lista)

        # 4. Exibir o resultado
    print("A lista de participantes é:", nomes_lista)
    print("---")
    print(f"O nome escolhido aleatoriamente foi: {nome_escolhido}")

else:
    print("A lista de nomes está vazia. Por favor, adicione nomes para poder sortear.")
```

Python: Framework

Introdução: O que é um Framework para Linguagem Python?

Biblioteca – coleção de Ferramentas, Funções e módulos

Framework é uma coleção de módulos Python que fornece um conjunto de funcionalidades comuns que podem ser usadas como uma estrutura para a criação de softwares de qualquer tipo.

Os frameworks são projetados para simplificar o processo de desenvolvimento, fornecendo uma diretriz geral sobre como devemos criar software e abstraindo algumas das tarefas mais complexas ou repetitivas. Isso permite que você se concentre em escrever uma lógica exclusiva e personalizada para seus aplicativos, em vez de ter que construir tudo do zero.

Uma tarefa repetitiva seria o tratamento de solicitações HTTP. Como a maioria dos aplicativos da internet precisa lidar com esse tipo de solicitações, os desenvolvedores usam frameworks existentes que facilitam essa função em vez de escrever tudo do zero ou reutilizar o mesmo código em diferentes projetos.

Tipos de Frameworks Python

A linguagem Python tem uma variedade de frameworks disponíveis para diferentes tipos de desenvolvimento.

Tipo: Framework Full-Stak

Tipo: Microframework

Tipo: Frameworks Assíncronos

Tipo: Frameworks Full-Stack

Framework Full-Stack é um conjunto de ferramentas que fornece tudo o que um desenvolvedor precisa para criar um aplicativo (software) web completo do início ao fim.

Isso inclui uma maneira de criar o frontend como por exemplo, um sistema de templates e uma abordagem para exibir informações ao usuário, e o backend que inclui funcionalidades comuns, como a criação de registros do banco de dados, o tratamento de solicitações HTTP e o controle da segurança do software/aplicativo.

Tipo: Microframeworks

Microframework é um framewrok minimalista que fornece apenas os componentes essenciais necessários para criar algum tipo de aplicativo/software.

Foram projetados para serem leves e fácil de estender, tornando uma boa opção para projetos pequenos ou para desenvolvedores que desejam ter mais controle sobre seus códigos.

Tipo: Frameworks Assíncronos

Um Framewrok Assíncrono é projetado para lidar com a concorrência e paralelismo, permitindo que os desenvolvedores construam aplicativos que possam realizar múltiplas tarefas simultaneamente.

Obrigado

Wellington S. S. Silva

guitomw@outlook.com