```python
import torch
import os

# ============================================================
# CONFIGURAÇÕES INICIAIS
# ============================================================
script_dir = os.path.dirname(os.path.abspath(__file__))  # Pega o caminho completo de PARTE_1
FILE_PATH = os.path.join(script_dir, '..', 'machado.txt')  # Sobe um nível e acha machado.txt

BLOCK_SIZE = 8  # Tamanho do contexto (janela de tempo)
BATCH_SIZE = 4  # Quantas sequências processar em paralelo
TRAIN_SPLIT = 0.8  # 80% para treino, 20% para validação

# ============================================================
# 1. CARREGAMENTO DOS DADOS E VOCABULÁRIO
# ============================================================
def load_data(file_path):
    """
    Lê o arquivo, cria o vocabulário e converte o texto para tensores inteiros.
    Equivalente ao trecho em R que usa readr::read_file e stringr::str_split.
    """
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            text = f.read()
        print(f"Arquivo '{file_path}' carregado com sucesso.")
        # Equivalente ao stringr::str_sub(data, 100, 1000) apenas para visualização
        print(f"Amostra do texto: {text[100:200]}..."
```

```
27 )
28     except FileNotFoundError:
29         print(f"Arquivo '{file_path}' não encontrado
   . Usando texto de exemplo.")
30         text = "Aqui está um texto de exemplo para
   simular o funcionamento do código. " * 500
31 
32 
   # ------------------------------------------------
   ------------------------
33     # Construção do Vocabulário
34     # R: vocab <- sort(unique(stringr::str_split_1(
   data, "")))
35 
   # ------------------------------------------------
   ------------------------
36     chars = sorted(list(set(text)))
37     vocab_size = len(chars)
38     print(f"Tamanho do vocabulário: {vocab_size}")
39 
40 
   # ------------------------------------------------
   ------------------------
41     # Mapeamentos (Encoder/Decoder)
42     # R: vocab <- structure(seq_along(vocab), names
    = vocab) (cria índices nomeados)
43     # Python: Criamos dicionários explícitos para
   conversão char <-> int
44 
   # ------------------------------------------------
   ------------------------
45     stoi = {ch: i for i, ch in enumerate(chars)}  #
   String to Int
46     itos = {i: ch for i, ch in enumerate(chars)}  #
   Int to String
47 
48     # Função para codificar: string -> lista de
   inteiros
49     encode = lambda s: [stoi[c] for c in s]
50 
51     # Função para decodificar: lista de inteiros ->
```

```
51 string
52     decode = lambda l: ''.join([itos[i] for i in l])
53
54
   # ------------------------------------------------------------
55     # Criação do Tensor de Dados
56     # R: data <- stringr::str_split_1(data, "") ... depois acessa via índices
57     # Python: Convertemos o texto para um tensor LongTensor (inteiros)
58
   # ------------------------------------------------------------
59     data_tensor = torch.tensor(encode(text), dtype=torch.long)
60     print(f"Tamanho total do dataset (caracteres): {len(data_tensor)}")
61     print(f"Primeiros 18 tokens (caracteres): '{decode(data_tensor[:18].tolist())}'")
62
63     return data_tensor, vocab_size, encode, decode
64
65
66 # ============================================================
67 # 2. SEPARAÇÃO TREINO / VALIDAÇÃO
68 # ============================================================
69 # Carrega os dados
70 data, vocab_size, encode, decode = load_data(FILE_PATH)
71
72 # Define o ponto de corte (n no R refere-se ao length(data))
73 n = len(data)
74 n_train = int(n * TRAIN_SPLIT)
75
76 # Divide os dados em dois tensores distintos
77 train_data = data[:n_train]
78 val_data = data[n_train:]
```

```
79
80
81 # ================================================================
   ============================
82 # 3. FUNÇÃO GET_BATCH
83 # ================================================================
   ============================
84 def get_batch(split, block_size=BLOCK_SIZE,
   batch_size=BATCH_SIZE):
85     """
86     Gera um lote (batch) de entradas (x) e alvos (y
   ).
87     Args:
88         split (str): 'train' ou 'valid' (qualquer
   coisa diferente de train).
89         block_size (int): Tamanho do contexto.
90         batch_size (int): Número de sequências no
   batch.
91
92     Returns:
93         x (torch.Tensor): Tensor de entrada (
   batch_size, block_size)
94         y (torch.Tensor): Tensor alvo (batch_size,
   block_size)
95     """
96     # Seleciona o dataset correto
97     # R: if (split == "train") { sample... } else {
   sample... }
98     data_source = train_data if split == 'train'
   else val_data
99
100     # Gera índices aleatórios para o início de cada
   bloco
101     # R: sample.int(0.8*(n-block_size), batch_size)
102     # Python: torch.randint gera inteiros aleatórios
    até high (len(data_source) - block_size)
103     ix = torch.randint(len(data_source) - block_size
   , (batch_size,))
104
105     # Constrói o tensor X empilhando os pedaços do
   texto
```

```
106     # R: x <- vocab[data[batch_ids]]; dim(x) <- dim(
    batch_ids)
107     # Python: Para cada índice i em ix, pegamos o
    slice data[i : i+block_size]
108     x = torch.stack([data_source[i: i + block_size]
    for i in ix])
109 
110     # Constrói o tensor Y (alvo) deslocado em 1
    posição à direita
111     # R: y <- vocab[data[batch_ids + 1]]
112     y = torch.stack([data_source[i + 1: i +
    block_size + 1] for i in ix])
113 
114     # R: list(x = torch_tensor(x), y = torch_tensor(
    y))
115     # Python: No PyTorch, x e y já são tensores aqui
     se data_source for tensor.
116     # Mas garantimos que estejam no dispositivo
    correto (CPU/GPU) se necessário no futuro.
117     return x, y
118 
119 
120 # ==================================================
    ============================
121 # TESTE DA IMPLEMENTAÇÃO (Main)
122 # ==================================================
    ============================
123 if __name__ == "__main__":
124     torch.manual_seed(1337)  # Para
    reprodutibilidade, assim como set.seed no R
125 
126     print("\n--- Testando get_batch('train') ---")
127     xb, yb = get_batch('train')
128 
129     print("Shape de X:", xb.shape)  # Deve ser [4, 8
    ] (batch_size, block_size)
130     print("Shape de Y:", yb.shape)
131 
132     print("\nConteúdo de X (Inputs):")
133     print(xb)
134 
```

```
135     print("\nConteúdo de Y (Targets/Labels):")
136     print(yb)
137 
138     print("\n--- Verificação Visual (Decodificação
    ) ---")
139     # Vamos pegar a primeira linha do batch para
    conferir se Y é o deslocamento de X
140     primeira_linha_x = xb[0].tolist()
141     primeira_linha_y = yb[0].tolist()
142 
143     print(f"X[0] decodificado: '{decode(
    primeira_linha_x)}'")
144     print(f"Y[0] decodificado: '{decode(
    primeira_linha_y)}'")
145     print("Nota: Y deve ser X deslocado uma letra
    para frente (next token prediction).")
```