Profa. Nádia Félix
Projeto TJ:
February 18, 2025

Tarefa 3: Arquivos

Instruções sobre como redigir e enviar seu trabalho estão disponíveis na página do curso. O não cumprimento dessas instruções resultará na perda de pontos. Direcione quaisquer perguntas ao professor.

Leia isto primeiro.

1) Comece cedo! Se você encontrar dificuldades ou tiver dúvidas, é melhor identificar esses problemas com antecedência, e não nas horas anteriores ao prazo de entrega!
2) Caso tenha dúvidas ou encontre problemas, não envie e-mails diretamente ao professor. Publique no fórum de discussão para que seus colegas também possam se beneficiar caso tenham a mesma dúvida.
3) Certifique-se de fazer backup do seu trabalho! Recomenda-se, no mínimo, salvar em uma pasta do Dropbox ou, melhor ainda, usar o Git, que vale o esforço de aprender.

Questão 1. Noções básicas de arquivos: leitura e escrita

Este problema lhe dará alguma prática com os conceitos básicos de leitura e escrita de arquivos de texto em Python.

1) Escreva uma função reverse_lines, chamada reverse_lines(infile, outfile), onde infile e outfile são strings especificando nomes de arquivo. Chamar reverse_lines(infile, outfile) deve sobrescrever outfile para que seja uma cópia de infile, exceto que cada linha de infile é invertida. Então, por exemplo, se o arquivo infile tem o conteúdo:

   hello
   tacocat
   1 2 3 4 5

   então o arquivo outfile deve ser:

   olleh
   tacocat
   5 4 3 2 1

   Você pode assumir que cada linha de infile termina com uma nova linha. Da mesma forma, cada linha de outfile deve terminar com uma nova linha. Seu programa deve executar a verificação de tipo e gerar um erro apropriado no caso de infile ou outfile não serem strings. reverse_lines deve abrir infile para leitura e abrir outfile para escrita (e gerar um erro se qualquer uma dessas operações falhar). Seu programa deve sobrescrever outfile se ele já existir.

   Dica: Escrever uma função reverse_string que receba um único argumento string e retorne essa string, invertida.

   Segunda dica: tenha cuidado com as quebras de linha no final das linhas! Quando você lê uma linha da entrada, o que acontece com a quebra de linha no final? Leia a documentação do Python com cuidado e verifique seu código em casos de teste simples!
2) Há pelo menos duas maneiras de implementar reverse_lines:
   - Opção 1: Abra infile para leitura e outfile para escrita. Leia as linhas de infile uma de cada vez e salve-as nas entradas de uma lista, digamos, line_list. Então, quando terminarmos de ler infile, itere sobre as entradas de line_list, revertendo cada uma e gravando-as em outfile.
   - Opção 2: Abra infile para leitura e outfile para escrita. Leia as linhas de infile uma de cada vez e, cada vez que lermos uma linha, inverta-a e grave-a em outfile imediatamente.

   Há pelo menos um bom motivo para preferir a Opção 2 em vez da Opção 1. Qual é esse motivo? Dica: se infile for especialmente grande, o que acontece com a lista line_list? Defina duas novas funções, revlines_storage e revlines_direct, que tenham a mesma assinatura (ou seja, peguem os mesmos argumentos) e executem a mesma função que reverse_lines. revlines_storage deve seguir

o design na Opção 1 e revlines_direct deve seguir o design descrito na Opção 2. Se você seguir um desses designs em sua solução para reverse_lines, você está livre para copiar e colar seu código (ou, melhor ainda, apenas fazer sua nova função chamar sua função já escrita), mas certifique-se de que você ainda está definindo revlines_direct e revlines_storage. Não há necessidade de executar verificação de erros nessas duas funções.

3) Para comparar revlines_direct e revlines_storage, precisamos de um arquivo para testar. Escreva uma função generate_test_file que receba três argumentos: uma string filename, um inteiro não negativo nlines e outro inteiro não negativo nchars, nessa ordem. Sua função deve abrir filename para escrita e escrever nlines linhas de texto, com cada linha contendo nchars caracteres, cada um escolhido uniformemente ao acaso das letras minúsculas (a,b,c,...,z). Sua função deve gerar um erro apropriado no caso de filename não ser uma string, ou no caso de nlines ou nchars não ser um inteiro não negativo. Dica: você pode escolher um inteiro aleatório em um intervalo dado usando a função randint no módulo random[1]. Então você pode usar esse inteiro aleatório para indexar em string. ascii_lowercase, que é uma string que consiste nas letras minúsculas do alfabeto.
4) Agora, vamos usar o módulo de tempo do Python para comparar a rapidez de revlines_direct e revlines_storage. Escreva uma função chamada time_trial que recebe dois inteiros não negativos, nlines e nchars, como seus argumentos, e retorna uma 2-tuple de floats. Sua função deve:
   (a) Usar generate_test_file para gerar um arquivo de teste de nlines linhas de texto, cada uma contendo nchars caracteres aleatórios.
   (b) Executar revlines_storage no arquivo de teste, usando o módulo de tempo do Python para medir quanto tempo o programa levou.
   (c) Executar revlines_direct no arquivo de teste, usando o módulo de tempo do Python para medir quanto tempo o programa levou.
   (d) Retornar uma tupla (t1, t2) onde t1 é um float representando o número de segundos que levou para executar revlines_storage no arquivo de teste e t2 é um float representando o número de segundos que levou para executar revlines_direct no arquivo de teste.

   Dica: para cronometrar uma operação, você pode modificar o código a seguir (presumo que você tenha importado o módulo time com import time).

```
start_time = time.time()
do_operation()
end_time = time.time()
t1 = end_time - start_time
```

5) Use time_trial para comparar os tempos de execução de revlines_storage e revlines_direct. Ao escolher valores adequados de nlines e nchars, você deve ser capaz de ver uma diferença notável entre os dois tempos de execução. O que você acha que explicaria essa diferença? Nota: não há necessidade de fazer um gráfico ou algo assim para isso. Basta executar o programa para algumas escolhas diferentes de argumentos, descrever a diferença que você vê e tentar explicar a diferença. Nota também: você pode precisar definir os argumentos para serem bem grandes antes de ver uma diferença entre os dois métodos diferentes, especialmente em máquinas mais novas e rápidas ou se você tiver muitos outros processos em execução no seu computador (por exemplo, muitas guias do navegador abertas).

[1] Veja https://docs.python.org/3/library/random.html#random.randintfordocumentation.

Questão 2. Contagem de bigramas de palavras Vamos escrever uma função para contar bigramas de palavras. Ou seja, para cada par de palavras, digamos, gato e cachorro, queremos contar quantas vezes a palavra "gato" ocorreu imediatamente antes da palavra "cachorro". Representaremos esse bigrama por uma tupla, ('gato', 'cachorro'). Para nossos propósitos, ignoraremos todas as quebras de linha, pontuação e capitalização em nossa contagem. Então, como exemplo, o fragmento de poema,

> Half a league, half a league,
> Half a league onward,
> All in the valley of Death
> Rode the six hundred.

inclui os bigramas ('half', 'a') e ('a', 'league') ambos três vezes, o bigrama ('league', 'half') aparece duas vezes, enquanto o bigrama ('in', 'the') aparece apenas uma vez.

1) Escreva uma função count_bigrams_in_file que receba um nome de arquivo como seu único argumento. Sua função deve ler do arquivo fornecido e retornar um dicionário cujas chaves são bigramas (fornecidos na forma de tupla acima) e os valores são as contagens para esses bigramas. Novamente, sua função deve ignorar pontuação, espaços, novas linhas e capitalização. As strings em suas tuplas de chave devem ser minúsculas. Sua função deve usar uma instrução try-catch para gerar um erro com uma mensagem apropriada para alertar o usuário no caso de o arquivo fornecido não poder ser aberto e um erro diferente no caso de o argumento fornecido não ser uma string. Dica: você encontrará a função Python str.strip(), junto com as constantes de string definidas na documentação de string (https://docs.python.org/3/library/string.html), úteis para remover pontuação. Dica: tenha cuidado para verificar se sua função manipula novas linhas corretamente. Por exemplo, no poema acima, um dos bigramas ('league', 'half') abrange uma nova linha, mas deve ser contado mesmo assim. Nota: tenha cuidado para que sua função não conte acidentalmente a string vazia como uma palavra (este é um bug comum se você não for cuidadoso ao dividir o texto de entrada).