Atividade Avaliativa #01: Revisão

Observações:

1. Os programas deverão ser desenvolvidos em linguagem PYTHON;
2. Em todas as questões serão cobradas a criação de funções pelo aluno e o uso de exceções. Esses pontos serão critérios avaliativos, ou seja, o não uso implica em decréscimo na nota em cada questão não implementada utilizando tais recursos.

1. Elaborar um programa que siga as seguintes especificações:

a) O programa deverá solicitar 3 valores inteiros sendo que, deverão ser tratadas as exceções caso os valores informados não sejam inteiros válidos;

b) Uma vez informados os valores válidos, deverá ser chamada uma função (para fins de padronização iremos nomear essa função como ***gerar_lista***). Essa função terá as seguintes especificações:

i. Ela irá receber 3 argumentos: ***quantidade***, ***valor_minimo*** e ***valor_maximo***;

ii. Ela irá retornar ao programa 2 valores, o primeiro um valor booleano (***True*** ou ***False***) informando se a lista foi gerada corretamente e o segundo valor será a lista gerada propriamente dita (caso tenha sido gerada), caso contrário o segundo valor deverá ser ***None***;

iii. A lista a ser gerada deverá conter a quantidade de valores informados no argumento ***quantidade***, Esses valores deverão estar compreendidos entre o argumento ***valor_minimo*** e ***valor_maximo***, não deverá ser ordenada e poderá conter repetições.

c) Uma vez que a lista tenha sido gerada, deverá ser chamada uma função (para fins de padronização iremos nomear essa função como ***salvar_lista***). Essa função terá as seguintes especificações:

i. Ela irá receber 2 argumentos: ***nome_lista***, ***nome_arquivo***;

ii. Ela irá retornar ao programa 1 valor booleano (***True*** ou ***False***) informando se a lista foi salva corretamente ou não.

iii. A lista informada no argumento ***nome_lista*** deverá ser salva no mesmo diretório/pasta da aplicação em um arquivo com nome informado no argumento ***nome_arquivo***;

iv. Cada linha do arquivo deverá conter apenas um valor da lista informada.

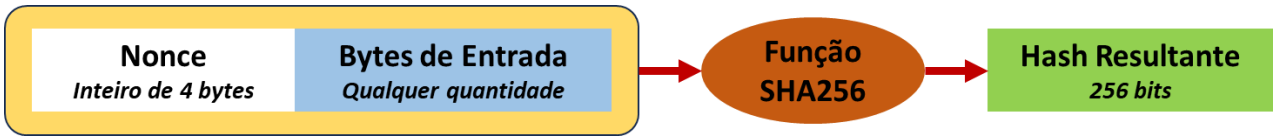
2. Elaborar um programa que siga as seguintes especificações:

a) O programa deverá solicitar o nome do arquivo salvo na questão 1 dessa lista de exercícios.
   i. Deverão ser tratadas as exceções que eventualmente possam surgir (atentem para caso o arquivo não exista).

b) Uma vez que o nome do arquivo seja válido, deverá ser chamada uma função (para fins de padronização iremos nomear essa função como ***ler_arquivo***). Essa função terá as seguintes especificações:
   i. Ela irá receber 1 argumento: ***nome_arquivo***;
   ii. Ela irá retornar ao programa 2 valores. O primeiro será um valor booleano (***True*** ou ***False***) informando se o arquivo foi lido com sucesso ou não e o segundo valor será a lista gerada com base no arquivo lido. Caso o primeiro valor de retorno seja ***False***, o segundo valor de retorno deverá ser ***None***.

c) Uma vez que já temos a lista lida do arquivo, deverá ser chamada uma função (para fins de padronização iremos nomear essa função como ***ordena_lista***). Essa função terá as seguintes especificações:
   i. Ela irá receber 2 argumentos: ***nome_lista*** e ***método_ordena***;
   ii. Ela irá retornar ao programa 2 valores. O primeiro será um valor booleano (***True*** ou ***False***) informando se a lista foi ordenada ou não e o segundo valor será a lista ordenada com base no método de ordenação informado no segundo argumento. Caso o primeiro valor de retorno seja ***False***, o segundo valor de retorno deverá ser ***None***;
   iii. O segundo argumento irá receber uma string informando o método de ordenação. As strings válidas para os métodos de ordenação são: ***BUBBLE***, ***INSERTION***, ***SELECTION*** e ***QUICK***;
   iv. Para cada um dos métodos de ordenação deverá ser criada uma função correspondente (para fins de padronização, cada função será nomeada da seguinte maneira: ***ordena_bubble***, ***ordena_insertion***, ***ordena_selection***, ***ordena_quick***) que deverá implementar o método de ordenação;
   v. Cada uma das funções de ordenação irá retornar ao programa 2 valores. O primeiro será um valor booleano (***True*** ou ***False***) informando se foi ordenada ou não e o segundo valor será a lista ordenada. Caso o primeiro valor de retorno seja ***False***, o segundo valor de retorno deverá ser ***None***.

3. A dificuldade de minerar bitcoins envolve ocorre porque é necessário executar o que se chama de prova de trabalho. Em outras palavras, vários mineradores competem para realizar uma tarefa; aquele que primeiro realizar é o minerador campeão da atividade e recebe uma boa recompensa. Na prática, a atividade a realizar é: receber um conjunto de transações (um conjunto de bytes) e calcular o hash SHA-256 deles, mas tem um detalhe: um número de quatro bytes deve ser adicionado no início dos bytes recebidos (chame-o de nonce) e dos 256 bits de resultado uma determinada quantidade inicial deve ser zero. O minerador que descobrir o nonce certo é o vencedor. Graficamente:



Portanto, minerar é: a) escolher um nonce; b) juntar com os bytes da entrada; c) calcular o hash desse conjunto; d) verificar se o hash resultante inicia com uma certa quantidade de bits em zero; e) se o hash calculado não atende ao requisito, repetir o processo.

Faça uma função em Python de nome findNonce que recebe três argumentos:

- **dataToHash** – um conjunto de bytes
- **bitsToBeZero** – o número de bits iniciais que deve ser zero no hash

e devolve:

- o **nonce** encontrado
- o **tempo** (em segundos) que demorou para encontrar o nonce

Ao final, faça um programa que usa a função para preencher a seguinte tabela:

Texto a validar (converta para bytes antes de chamar)	Bits em zero	Nonce	Tempo (em s)
“Esse é fácil”	8		
“Esse é fácil”	10		
“Esse é fácil”	15		
“Texto maior muda o tempo?”	8		
“Texto maior muda o tempo?”	10		
“Texto maior muda o tempo?”	15		
“É possível calcular esse?”	18		
“É possível calcular esse?”	19		
“É possível calcular esse?”	20		

Sua resposta deve ser 3 arquivos: um arquivo com o programa principal, um segundo com a função e outras auxiliares, se necessário e o terceiro com a tabela preenchida (em formato doc, PDF ou txt).

4. Desenvolver um programa que simula o site **https://term.ooo/**, mas com a palavra do dia sendo sorteada a partir de um arquivo texto. A seguir tem o detalhamento das funcionalidades:

- ***Leitura do Arquivo:***
  - O programa deve ler um arquivo texto com uma lista de palavras.
  - Cada linha do arquivo deve conter uma palavra com no mínimo 5 e no máximo 8 letras.
  - O programa deve armazenar as palavras em uma lista.
- ***Sorteio da Palavra:***
  - O programa deve sortear uma palavra aleatória da lista de palavras.
  - O programa informa quantas letras a palavra sorteada tem.
- ***Jogo:***
  - O usuário tem 6 tentativas para adivinhar a palavra sorteada;
  - O usuário deve digitar uma palavra limitada a quantidade de letras que a palavra sorteada possui (tratar caso a quantidade de letras seja diferente);
  - Se a palavra for válida (possuir a mesma quantidade de caracteres da palavra sorteada), o programa deve fornecer feedback sobre a tentativa:
    - Para cada letra:
      - ✓ Se a letra estiver na posição correta, a cor da letra deve ficar verde.
      - ✓ Se a letra estiver presente na palavra, mas em posição incorreta, a cor da letra deve ficar amarela.
      - ✓ Se a letra não estiver presente na palavra, a cor da letra deve ficar cinza.
  - O usuário pode tentar adivinhar a palavra novamente após cada tentativa.
- ***Vitória ou Derrota:***
  - Se o usuário adivinhar a palavra em 6 tentativas ou menos, o programa deve parabenizá-lo e mostrar o número de tentativas utilizadas.
  - Se o usuário não conseguir adivinhar a palavra em 6 tentativas, o programa deve revelar a palavra e informar que o usuário perdeu.

5. Faça um programa que lê três parâmetros:

a) Nome de um arquivo origem;

b) Uma palavra-passe;

c) Nome de um arquivo destino.

Sobre cada um dos bytes do arquivo de origem aplica uma operação de ‘xor’, considerando o valor ASCII das letras da palavra-passe. Assim, se a palavra-passe for 'pato', o primeiro byte do arquivo destino é o ‘xor’ do respectivo primeiro byte do arquivo de origem com o código ASCII do 'p'; o segundo byte usa o código ASCII do ‘a’ para gerar o segundo byte do arquivo de destino, a partir do segundo do arquivo de origem. E assim sucessivamente. Após o uso do último byte da palavra-passe, volte a usar o primeiro e o processo se repete.

Não esqueça de tratar as exceções. Não sobrescreva arquivos existentes, notifique o usuário nesses casos. Você deve entregar somente o programa (com comentários).