Lista De Exercícios #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. Implemente uma calculadora de sub-rede em Python. O programa deve solicitar um endereço IP, uma máscara de rede inicial e uma máscara de rede final. Para cada máscara de rede no intervalo especificado, o programa deve calcular e exibir as seguintes informações:

a) Endereço de Rede
b) Primeiro Host
c) Último Host
d) Endereço de Broadcast
e) Máscara de Sub-rede em Decimal e Binário
f) Número de Hosts Válidos.

Requisitos:

a) **NÃO** utilize a biblioteca ipaddress;
b) Valide o endereço IP e as máscaras de rede fornecidas pelo usuário;
c) Salve os resultados em um arquivo no formato JSON (dicionário). Não subscreva arquivos existentes;
d) Formate as saídas de forma clara e organizada (se quiser pode usar a biblioteca tabulate).

Exemplo de entrada de dados:

```
Digite o endereço IP: 10.1.0.0
Digite a máscara de rede inicial (CIDR): 8
Digite a máscara de rede final (CIDR): 10
```

Exemplo de saída de dados:

```
Para máscara /8:
Endereço de Rede: 10.0.0.0
Primeiro Host: 10.0.0.1
Último Host: 10.255.255.254
Endereço de Broadcast: 10.255.255.255
Máscara de Sub-rede: 255.0.0.0
Máscara de Sub-rede (binário): 11111111.00000000.00000000.00000000
Número de Hosts Válidos: 16777214

Para máscara /9:
Endereço de Rede: 10.0.0.0
Primeiro Host: 10.0.0.1
Último Host: 10.127.255.254
Endereço de Broadcast: 10.127.255.255
Máscara de Sub-rede: 255.128.0.0
Máscara de Sub-rede (binário): 11111111.10000000.00000000.00000000
Número de Hosts Válidos: 8388606

Para máscara /10:
Endereço de Rede: 10.0.0.0
Primeiro Host: 10.0.0.1
Último Host: 10.63.255.254
Endereço de Broadcast: 10.63.255.255
Máscara de Sub-rede: 255.192.0.0
Máscara de Sub-rede (binário): 11111111.11000000.00000000.00000000
Número de Hosts Válidos: 4194302
```

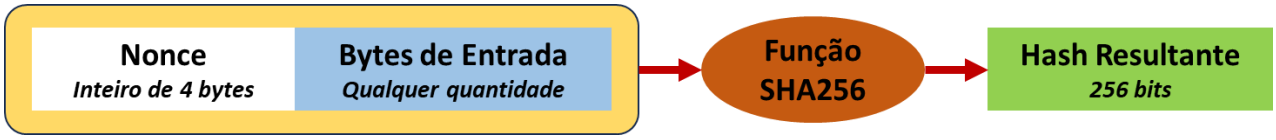
Exemplo de saída de dados (usando a biblioteca tabulate):

CIDR	Network Address	Primeiro Host	Último Host	Broadcast Address	Subnet Mask	Subnet Mask (Binary)	Hosts Válidos
/8	10.0.0.0	10.0.0.1	10.255.255.254	10.255.255.255	255.0.0.0	11111111.00000000.00000000.00000000	16777214
/9	10.0.0.0	10.0.0.1	10.127.255.254	10.127.255.255	255.128.0.0	11111111.10000000.00000000.00000000	8388606
/10	10.0.0.0	10.0.0.1	10.63.255.254	10.63.255.255	255.192.0.0	11111111.11000000.00000000.00000000	4194302

Exemplo do conteúdo do arquivo JSON:


```
[
    {
        "CIDR": "/8",
        "Endereco de Rede": "10.0.0.0",
        "Primeiro Host": "10.0.0.1",
        "Ultimo Host": "10.255.255.254",
        "Endereco de Broadcast": "10.255.255.255",
        "Mascara de Sub-Rede": "255.0.0.0",
        "Mascara de Sub-Rede (Binario)": "11111111.00000000.00000000.00000000",
        "Hosts Validos": 16777214
    },
    {
        "CIDR": "/9",
        "Endereco de Rede": "10.0.0.0",
        "Primeiro Host": "10.0.0.1",
        "Ultimo Host": "10.127.255.254",
        "Endereco de Broadcast": "10.127.255.255",
        "Mascara de Sub-Rede": "255.128.0.0",
        "Mascara de Sub-Rede (Binario)": "11111111.10000000.00000000.00000000",
        "Hosts Validos": 8388606
    },
    {
        "CIDR": "/10",
        "Endereco de Rede": "10.0.0.0",
        "Primeiro Host": "10.0.0.1",
        "Ultimo Host": "10.63.255.254",
        "Endereco de Broadcast": "10.63.255.255",
        "Mascara de Sub-Rede": "255.192.0.0",
        "Mascara de Sub-Rede (Binario)": "11111111.11000000.00000000.00000000",
        "Hosts Validos": 4194302
    }
]
```

2. 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 2 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).

3. 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.

4. 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).