Checkpoint 2 – 1TDSPA 2024

Computational Thinking with Python - 05/04/2024

Instruções:

- As questões possuem o mesmo peso (1)
- Crie 1 (ou mais) arquivo .py e o envie pelo Teams para o professor
    - Caso modularize o código e utilize imports, não se esqueça de zipar
- Faça cópias de segurança (backup) de suas soluções intermediárias
- Teste seu código
- Caso utilize alguma ferramenta de consulta, entenda o que recebeu de dicas e não simplesmente copie e cole
- Evite soluções muito complexas
- Entregas similares serão desconsideradas, ou seja, não copie a solução de seu colega
- Você terá até dia 12/04 às 11h59 para enviar sua solução.
- Utilize funções sempre que possível
- Peça login e senha sempre que puder

Questão 1: Validação de login e senha

A forma mais básica de autenticação de usuários é por meio dos pares de **login** e **senha**. O login pode ser um e-mail, nome de usuário ou algum documento, como o CPF. Já a senha, pode possuir alguns requisitos como tamanho mínimo e quantidade mínima de alguns caracteres, como alfabéticos, maiúsculas, números, etc. Tendo isso em mente, **implemente uma função** em Python que crie um login e senha para um usuário, **recebendo** e **validando** os dados obtidos por meio de inputs.

Requisitos:

- Deverá necessariamente implementar uma função
- Login: ser um dos 3 tipos a seguir
    - email no formato: xxxxx@yyy.zzz
    - user-name: apenas conter caracteres alfabéticos e numéricos. A única exceção para caractere especial é o caractere “_”
    - documentos:

        - CPF: XXX.XXX.XXX-XX ou XXXXXXXXXXX
        - RG: XX.XXX.XXX-X ou XXXXXXXXX
    - Senha: deverá conter **ao menos 12** caracteres e ser composta **por pelo menos 2** caracteres de cada tipo
        - Numérico: 0-9
        - Alfabético maiúsculo: A-Z
        - Alfabético minúsculo: a-z
        - Especiais: !@#$%&*()[]{};,.:/\|

Dicas:

- Não utilize expressões regulares, a menos que você saiba explicá-las
- Note que nem todos os caracteres especiais serão aceitos, apenas os listados nos requisitos
- Pergunte ao usuário qual o tipo de seu login, isso facilitará a validação

Questão 2: Cadastrando usuários

Utilizando a rotina de validação da questão 1, vamos agora cadastrar um usuário, sua senha e sua "role". Para isso, **utilize uma lista de dicionários**, em que cada dicionário conterá as credenciais de um usuário. As chaves de cada dicionário serão "login", "senha" e "role", sendo que esta última estará atrelada ao tipo de usuário.

Na sequência, crie um menu em que a primeira opção seja cadastrar um usuário. A opção de cadastro **deverá pedir o login e a senha** para validar o usuário que está tentando cadastrar e também só deverá permitir que usuários do tipo "admin" realizem o cadastro. Caso positivo, peça o nome de usuário, senha e role, sempre validando os dados inseridos. Caso contrário, lance uma exceção deixando nítida a tentativa de "ataque" a seu sistema.

As roles existentes são apenas: *admin* e *user*, sendo que um admin possui todos os privilégios de um user. Caso uma role inválida seja inserida, você deverá avisar ao usuário e coletar a nova role.

Requisitos:

- reutilizar a função da Questão 1 (sem duplicá-la) para validar os novos login e senha.
- Utilize funções

Questão 3: Atualizando o login e senha

Ofereça a opção de atualizar o login e a senha cadastrados na questão 2. Note que apenas o próprio usuário poderá alterar sua senha, mas um usuário admin poderá alterar a senha de qualquer outro usuário, menos de outro admin.

Questão 4: Encontre um número primo

Crie uma função que dado um número inteiro N, encontre o maior número primo até N. Lembre-se de ler o número N como input do usuário, procurar o primo e em seguida imprimir o número encontrado.

Requisitos:

- N deverá ser maior ou igual a 2.
- Insira essa função como uma opção do menu
- Além de imprimir o primo, **retorne-o**

Dica:

- O número primo poderá ser menor ou igual a N
- Iterar em ordem reversa

Questão 5: Protegendo a senha

Um passo básico para aumentar a segurança de uma aplicação é proteger as senhas salvas no banco de dados, utilizando criptografia. Desse modo, caso um agente mal-intencionado ganhe acesso ao banco, as senhas não poderão ser utilizadas por ele. Pensando nisso, vamos proteger a senha que nosso usuário cadastrou utilizando criptografia.

Nosso **primeiro passo** será converter todos os caracteres da senha para uma representação numérica, utilizando a função *ord*. Para isso, **crie uma função** que itere sobre todos os caracteres, e a cada um aplique a função ord com o objetivo de concatenar todos os números obtidos. Por exemplo:

```
for c in senha:
    senha_ord += str(ord(c))
```

Na sequência, converta a string encontrada para um inteiro e faça seu *hash*, criptografando a senha dada. O modo mais simples de se criar um *hash*, é utilizar divisões de módulo por um número primo. Para isso, utilizaremos o resultado da questão 4, quando calculamos um número primo menor que N. Tal inteiro "grande" será nossa "chave" criptográfica, de modo que apenas com ela poderemos validar a nova senha.

Logo, os passos de nosso algoritmo serão:

- Ler e validar a senha (questão 2)
- Converter a senha para representação numérica, utilizando a função *ord*
- Calcular o *hash* da senha com *int(senha_ord) % num_primo*
- Sobrescrever a senha original com o valor do *hash* e não mais o valor inserido pelo usuário

Deste modo, caso um hacker obtenha acesso a nosso código, ele não saberá qual a senha real de nosso usuário.

Você deverá **criar uma função chamada hash**, que faça tal conversão e na sequência, aplicá-la a todas as senhas salvas em seu "banco de dados" (lista de dicionários da questão 2)

Dicas:

- Os exemplos de código aqui não funcionarão simplesmente copiando para o arquivo .py, entenda-os e utilize como base.
- Utilize a função que calcula o número primo menor que N para a função de *hash*.
- Salve em cada um dos dicionários qual foi o número primo utilizado, assim

Questão 6: Login com a senha protegida

Agora que temos um código mais protegido, você deverá alterar os passos de login para que quando o usuário digitar sua senha, você faça a operação de hash e compare os hashes ao invés de comparar as senhas diretamente.

Questão 7: Persistindo o banco de dados

Tendo um código com as senhas protegidas, agora é a hora de persistirmos os nossos dados de forma segura. Para isso, iremos salvar os dados dos usuários em um arquivo JSON, simulando um banco de dados, como fizemos em aula. Deste modo, sempre que iniciarmos o programa ou criarmos uma entrada, armazenaremos o resultado no disco.

Questão 8: Encontrando um id para o usuário

Vamos aqui repetir o processo que vimos em sala, quando utilizamos o documento do usuário para encontrar seu id. Em sala, utilizamos o CPF do usuário, mas agora utilizaremos uma combinação de LOGIN+SENHA_HASHED para calcular esse id. O user_id também deverá ser salvo na lista de dicionários e no banco de dados em formato JSON.

Questão 9: Encontrando o pokémon do usuário

Crie uma função que acesse a Poke API e recupere os dados de pokémon referentes ao id do usuário. Você deverá criar uma nova chave no dicionário do usuário, chamada "poke_human", que conterá um novo dicionário, com as chaves:

- name: str com o nome do pokémon
- abilities: lista de dicionários (1 para cada habilidade), sendo que cada dicionário de habilidades conterá as seguintes chaves:
    - nomes: uma lista de tuplas com os nomes em cada idioma e o código do idioma
    - efeitos: lista de strings com cada efeito
    - flavors: lista de strings com cada flavor
    - lista de cada pokemon que possui a mesma habilidade

Questão 10: Atualizando o banco de dados

Agora que já atualizamos a lista de dados em memória, não se esqueça de que as adições da entrada Pokemon deverão ser refletidas no banco de dados que está sendo salvo no disco, em formato JSON.