INSTITUTO FEDERAL Rio Grande do Norte	**INSTITUTO FEDERAL DO RN** **Campus Natal-Central**
	Disciplina: Teste de Software
	Professor(a): Plácido A. Souza Neto
	Discente: **Matrícula:**
	Curso: TADS **Semestre:** 2024.1
	Prova 01

1. Validação de Senha é uma terefa de extrema importância no desenvolvimento de sistemas. É importante sempre estabelecer critérios de validação de senha. Por exemplo, uma senha pode ser considerada válida se:

   - **Tamanho**: Entre 8 e 20 caracteres.
   - **Complexidade**: Conter pelo menos uma letra maiúscula, uma minúscula, um número e um caractere especial.

   Classes de Equivalência As classes de equivalência dividem os possíveis valores de entrada em grupos que esperamos que produzam o mesmo resultado. Para a validação de senha, defina pelo menos 8 classes de equivalência para uma função de validação de senha. Atribua um nome para cada classe, e a partir das classes, construa casos de teste para cada uma das classes utilizando TDD (respeitando o formato *test_nome_classe_equivalencia*).

```
import unittest

def validar_senha(senha):
    # Implementação da lógica de validação aqui
    # ...

class TestValidacaoSenha(unittest.TestCase):
    def test_classe1(self):
        self.assertTrue(validar_senha("Ab123@"))

    def test_classe2(self):
        self.assertFalse(validar_senha("123456"))

    # ... outros testes para as demais classes de equivalência ...

    def test_classe8(self):
        self.assertFalse(validar_senha("123456"))
```

2. Imagine um sistema de gerenciamento de sirene de alarme do campus Natal-Central. Poderíamos utilizar Objetos Mock para testar esse programa. O gerenciamento da sirene deve considerar que esta deve tocar nos momentos de mudanças de horário de aulas e intervalos. Implemente os testes utilizando Objetos Mock para testar o método *tocar_sirene()* da classe *Sirene* que representa a sirene do campus.

   Estabeleça os testes considerando os **valores limites e as classes de equivalencia** de horários de aulas e horários de intervalo para os turnos Matutino e Vespertino.

Implementação das classes

```
class Sirene:
    def tocar(self):
        #implementacao da sirene tocando

class Relogio:
    def hora_atual(self):
        #implementacao da hora atual

class HorarioAulaOuIntervalo:
    def proximo_evento(self):
        #implementacao do proximo evento
```

Implementação da classe GerenciadorSirene

```
class GerenciadorSirene:
    def __init__(self, sirene: Sirene, relogio: Relogio, horario: HorarioAulaOuInterva
        self.sirene = sirene
        self.relogio = relogio
        self.horario = horario

    def verificar_eventos(self):
        hora_atual = self.relogio.hora_atual()
        proximo_evento = self.horario.proximo_evento()
        if hora_atual == proximo_evento.hora:
            self.sirene.tocar()
```

Implementação dos testes Mock

```
from unittest.mock import Mock

class TestGerenciadorSirene(unittest.TestCase):
    def test_tocar_sirene_no_horario1_manha(self):
        sirene = Mock()
        relogio = Mock()
        relogio.hora_atual.return_value = "7:00"
        # ... outros mocks para os demais objetos ...

    def test_tocar_sirene_no_intevalo_manha(self):
        ...
        relogio.hora_atual.return_value = "8:30"
        # ... outros mocks para os demais objetos ...

    # outras funções de teste para os valores
    # limites de horários de aulas e intervalos
```

OBS: Cada membro da dupla deve implementar a metados dos casos de teste e metade das operações. Quando 1 membro implementa um caso de teste, o outro implementa a função, seguindo a metodologia do TDD para a 1a questão. Essa abordagem será verificada a partir dos commits no repositório. Assim, é sempre necessário realizar primeiro o commit com o teste (por um dos membros), e depois o commit da implementação (por outro membro)