Organizando uma eleição

O último ano foi extremamente desafiador para você. Além de estarmos vivendo uma pandemia que aumentou nossa preocupação com a nossa própria vida e com nossos entes queridos e estarmos obrigados a passar mais tempo em casa para garantir nossa segurança, você se matriculou em um curso de programação e ciência de dados para mudar de rumo em sua carreira!

O *timing* pareceu ideal: mais tempo em casa, mais tempo para estudar. Porém, você esbarrou em algumas dificuldades. O condomínio está em reforma, e você passa o dia ao som de marretadas no piso. Durante a noite, no horário da sua aula online, seus vizinhos - um casal passando por dificuldades na relação - costumam discutir em voz alta. A discussão deles frequentemente acorda o bebê do outro vizinho e todos os cachorros do andar no prédio. Deixar para estudar no final de semana não ajuda, pois o filho adolescente da família do apartamento logo acima do seu costuma trazer os colegas de banda para ensaiar.

A situação se tornou insustentável, e o síndico - que é o mesmo senhorzinho desde que você se mudou para esse condomínio, há mais de 10 anos - não toma atitudes e não gosta de ser perturbado. Chegou a hora de organizar uma eleição para um novo síndico, e você tomou a iniciativa de montar um sistema eletrônico para auxiliar na votação.

Entidades envolvidas

Podemos imaginar as seguintes entidades envolvidas na eleição:

- **Morador**: representa cada uma das pessoas que moram no prédio. Possui um nome, um Apartamento e é capaz de depositar um voto em um Candidato em uma Urna.
- **Candidato**: é um tipo especial de Morador. Possui tudo o que o morador possui, mas também possui um número que será usado para representá-lo na Urna e uma contagem individual de votos.
- **Apartamento**: representa cada uma das unidades do prédio. Possui uma lista de moradores e um status indicando se ele já votou ou não. Cada apartamento tem direito a exatamente 1 voto. Se a sua filha já votou, você não pode mais votar!
- **Urna**: coleciona votos. Ela possui uma coleção de candidatos e uma coleção de moradores (lembrando que o candidato também conta como morador!). Ela armazena a quantidade de votos de cada candidato e é capaz de incrementar essa quantidade ao receber um voto novo. Ela também possui uma coleção de apartamentos e é capaz de determinar quais apartamentos já votaram (mas não qual apartamento votou em cada candidato - o voto é secreto). Quando o último apartamento votar, ela publica os resultados e declara o vencedor.

Morador

A classe mais básica de nossa eleição, representando cada uma das pessoas que habitam no condomínio. O morador obrigatoriamente possui um nome e um Apartamento.

Seu construtor pode ou não receber uma string contendo o nome, e pode ou não receber um objeto Apartamento. Caso ele não receba um nome, deverá solicitar a digitação de um através do input. Caso ele não receba um objeto Apartamento, deverá pedir por input o número do apartamento e criar o objeto.

O nome e apartamento digitados ou recebidos deverão se tornar atributos do objeto.

Nosso objeto também deverá possuir um método para votar. Ele deverá receber um objeto Urna e pode receber um número inteiro correspondente ao número de um candidato ou, caso não receba, irá ler o número pelo teclado. Ela deverá primeiro verificar se o seu apartamento já votou, e caso não tenha, ela irá chamar o método de acrescentar voto na Urna, informando o número de seu candidato e passando seu objeto apartamento. Caso contrário, ela exibirá uma mensagem alertando que esse apartamento já votou.

Candidato

Objetos da classe Candidato são, necessariamente, moradores. Eles possuem as mesmas informações e as mesmas ações que um morador. O que isso significa em programação orientada a objeto?

Ele possui outros dois atributos: o seu número que será usado para identificá-lo na Urna e sua contagem de votos. Ambos devem ser inicializados com valor 0.

Ele deve oferecer métodos para que seu número seja configurado (quando cadastrarmos o candidato na Urna, ela irá fornecer um número para ele), bem como para que sua contagem de votos seja atualizada (ao final da votação, a Urna irá atualizar a contagem de todos os candidatos).

Apartamento

Objetos da classe Apartamento irão agregar objetos Morador. Isso significa que um de seus atributos será uma lista de moradores. Quando um Apartamento é criado, sua lista começará vazia. O Apartamento deve oferecer métodos para adicionar moradores novos (recebe um objeto Morador e adiciona à lista) e para visualizar seus moradores.

Cada apartamento tem direito a exatamente um voto na eleição para síndico. Portanto, cada objeto também deve ter um atributo indicando se aquele apartamento já votou ou não. A classe deve fornecer meios para que a esse atributo possa ser atualizado, pois quando um morador votar, a urna irá alterar esse atributo registrando que o apartamento já votou.

Urna

A classe mais complexa do projeto. Ela é responsável por boa parte da lógica da nossa eleição.

Seu construtor deverá criar como atributos uma coleção de apartamentos e outra de candidatos - escolha a estrutura de dados que julgar mais adequada.

A classe deve fornecer métodos que permitam o cadastro de apartamentos (recebendo como parâmetro objetos Apartamento já existentes) e candidatos (idem). Ao receber um candidato para cadastro, ela deverá gerar um número **único** para representar esse candidato na eleição e irá ajustar esse atributo diretamente no objeto candidato.

Ela também deve ter um método para receber um voto, que deve receber um objeto apartamento e o número de um candidato. Ela deve atualizar a contagem de voto do candidato cujo número foi recebido, e marcar aquele apartamento como já votado. Se o apartamento que votou era o último que ainda podia votar, a urna deve imediatamente imprimir os resultados.

Bole uma maneira que a urna possa sinalizar para o programa principal se ela já encerrou a votação ou não.

O programa principal

Seu programa principal terá 3 etapas - fique à vontade para modularizar essas etapas com o auxílio de funções, ou mesmo utilizar outras classes se julgar necessário.

- Cadastro: o programa deverá ler informações sobre cada pessoa e ir criando os objetos Morador/Candidato e Apartamento correspondentes. Lembre-se de sempre perguntar se a próxima pessoa a ser lida é ou não candidata para decidir a melhor forma de instanciá-la no sistema.
- Configuração: o programa deverá utilizar as informações obtidasna etapa anterior para criar uma urna.
- Votação: o programa ficará em loop permitindo que diferentes moradores votem (ou tentem votar). Nessa etapa, o programa deve ser capaz de identificar qual o morador votando e utilizar os métodos criados previamente para realizar o voto. Quando a classe Urna informar que a votação acabou, o loop deve ser encerrado.

In [1]:
```
# ---------------------------------------------------------------------------
# Let's Code
# Projeto 2 - Sistema para votação de síndico
# por Alexandre Rozante
#
import csv
import time
import matplotlib.pyplot as plt

# ---------------------------------------------------------------------------
# Exceções
class ErroApartamento(Exception):
    pass

class ErroMorador(Exception):
    pass

class ErroCandidato(Exception):
    pass

class ErroMoradores(Exception):
    pass

class ErroUrna(Exception):
    pass

class ErroMenu(Exception):
    pass

# ---------------------------------------------------------------------------
# Classe Apartamento
#
# Representa um apartamento contendo moradores
#
# Atributos:
# - Número do apartamento
# - Lista de moradores
# - Flag de status de votação (True ou False)
#
class Apartamento:
```

```
    # Construtor
    def __init__(self, numero):
        if type(numero) != int:
            raise ErroApartamento('Um parâmetro tipo int é esperado.')
        self._numero = numero      # Número do apartamento
        self._moradores = []       # Lista de moradores
        self._votou = False        # Status de votação

    # Número do apartamento
    @property
    def numero(self):
        return self._numero

    # Quantidade de moradores
    @property
    def qtdMoradores(self):
        return self._moradores.__len__()

    # Exibe os moradores
    @property
    def strMoradores(self):
        s = 'Moradores:\n'
        for i in range(self.qtdMoradores):
            s += '   {}) {}\n'.format((i + 1), self._moradores[i].nome)
        return s

    # Retorna um morador pelo seu índice na lista
    def morador(self, indice):
        return self._moradores[indice]

    # Status de votação
    @property
    def votou(self):
        return self._votou

    @votou.setter
    def votou(self, votou):
        if type(votou) != bool:
            raise ErroApartamento('Um parâmetro bool é esperado.')
        self._votou = votou

    # Adição de Morador ao Apartamento
    def adicionarMorador(self, morador):
        nomeNovoMorador = morador.nome.lower().strip(' ')
        for moradorExistente in self._moradores:
            if moradorExistente.nome.lower().strip(' ') == nomeNovoMorador:
                return False
        self._moradores.append(morador)
        return True

    # Exclusão de Morador
    def excluirMorador(self, morador):
```

```
        self._moradores.remove(morador)

    # Exibe os moradores
    def exibirMoradores(self):
        print(self.strMoradores)

    # Representação visual do objeto
    def __repr__(self):
        return 'Apto={}\n'.format(self.numero) + self.strMoradores


# ------------------------------------------------------------------------
# Classe Morador
#
# Representa um morador de um apartamento
#
# Atributos:
# - Nome do obter_morador
# - Apartamento em que reside (objeto)
#
# Caso o nome não seja informado o método solicita sua digitação.
#
# Caso o apartamento não seja informado o método solicita seu número e instancia
# um novo objeto Apartamento.
#
class Morador:

    # Construtor
    def __init__(self, nome='', apto=None):
        # Apesar do nome ser opcional, Python permite que valores de diferentes tipos sejam passados.
        # Verificamos se é um valor String, e caso negativo, forçamos digitação ignorando o valor informado.
        if type(nome) != str:
            raise ErroMorador('Tipo inválido para o parâmetro "nome" do objeto Morador')

        # Procedimento similar quanto ao tipo de dado do parâmetro apto (Apartamento)
        if apto is not None and type(apto) != Apartamento:
            raise ErroMorador('Tipo inválido para o parâmetro "apto" do objeto Morador')

        # Nome não especificado? Solicita digitação
        while len(nome.strip(' ')) < 1:
            # Ao menos um caractere para o nome do morador deve ser digitado
            nome = input('Favor digitar o nome....: ')

        # Apartamento não especificado?
        if apto is None:
            # Solicita um número de apartamento maior que zero
            num_apto = 0
            while num_apto < 1:
                try:
                    num_apto = int(input('Número do apartamento...: '))
                except ValueError:
                    print('Número inválido.')
```

```
            # Instancia um novo apartamento para o obter_morador
            apto = Apartamento(num_apto)

        self._nome = nome
        self._apartamento = apto

    # Nome do morador
    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, novo_nome):
        self._nome = novo_nome

    # Apartamento
    @property
    def apartamento(self):
        return self._apartamento

    @apartamento.setter
    def apartamento(self, apto):
        self._apartamento = apto

    # Registro de voto do morador
    def votar(self, urna, numCandidato=0):
        interativo = False

        # Se não for especificada uma urna valida aborta o método
        if not isinstance(urna, Urna):
            raise ErroMorador('Objeto urna inválido')

        # Se um valor não numérico ou não inteiro for passado como número de candidato, força digitação
        if type(numCandidato) != int:
            raise ErroMorador('Número do candidato deve ser int.')

        # Se o número de candidato informado não existe, força digitação
        if numCandidato > 0:
            if not urna.numCandidatoValido(numCandidato):
                raise ErroMorador('Candidato inexistente {}'.format(numCandidato))
        else:
            # Enquanto um número válido de candidato não for especificado continua solicitando
            while numCandidato < 1:
                interativo = True
                # Apresenta a relação de candidatos e aguarda votação
                urna.exibirCandidatos()
                try:
                    numCandidato = int(input('Deseja votar em qual candidato? '))
                    if not urna.numCandidatoValido(numCandidato):
                        numCandidato = 0
                        raise ErroMorador()
                except Exception:
```

```
                    print('Candidato inválido')
        # Registra o voto realizado
        urna.registrarVoto(self.apartamento, numCandidato)
        if interativo:
            print('\n* * * Voto registrado com sucesso * * *.\n')

    # Representação visual do objeto
    def __repr__(self):
        return 'Morador {}, Apto.{}'.format(self.nome, self.apartamento.numero)


# -----------------------------------------------------------------------------
# Classe Candidato (extende Morador)
#
# Representa um morador que é um candidato a síndico
#
# Atributos:
# - Nome do obter_morador
# - Apartamento
# - Número de candidato
# - Votos recebidos
#
class Candidato(Morador):

    # Construtor
    def __init__(self, nome='', apto=None):
        super().__init__(nome, apto)
        self._numero = 0
        self._votos = 0

    # Número de candidato (r/w)
    @property
    def numero(self):
        return self._numero

    @numero.setter
    def numero(self, num):
        if type(num) != int:
            raise ErroCandidato('Parâmetro num deve ser um int')
        if num < 1:
            raise ErroCandidato('Parâmetro num deve ser um valor positivo maior que 0')
        self._numero = num

    # Votos recebidos é somente leitura.
    # Um método específico para incrementar unitariamente foi criado para maior segurança do processo de votação
    @property
    def votos(self):
        return self._votos

    # Registra um voto para o candidato
    def registrarVoto(self):
        self._votos += 1
```

```python
    # Representação visual do objeto
    def __repr__(self):
        reprMorador = super().__repr__()
        return 'Candidato {} - {} - Votos: {}'.format(self._numero, reprMorador, self._votos)


# -----------------------------------------------------------------------------
# Classe Moradores
#
# Armazena apartamentos, moradores e candidatos para um processo de eleição de
# síndico
#
# Atributos:
# - Dicionário de moradores (que podem ser candidatos ou não)
# - Dicionário de candidatos
# - Dicionário de apartamentos
#
class Moradores:

    # Construtor
    def __init__(self):
        self._limpar()

    # Limpeza para uma nova carga de dados
    # É utilizado no instanciamento de novos objetos e pelo método carregarCSV
    def _limpar(self):
        self._moradores = {}
        self._candidatos = {}
        self._apartamentos = {}

    # Retorna a quantidade de moradores cadastrados
    @property
    def qtdMoradores(self):
        return self._moradores.__len__()

    # Retorna uma representação string dos moradores
    @property
    def strMoradores(self):
        # Como podemos ter moradores homônimos em apartamentos distintos, logo informamos o apartamento de cada morador
        # É conveniente exibir em ordem crescente de nº de apartamento
        s = ''
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        for apto in aptos:
            s += '{}\n'.format(apto[1])
        return s

    # Retorna o dicionário de candidatos
    @property
    def candidatos(self):
        return self._candidatos
```

```python
    # Retorna a quantidade de candidatos cadastrados
    @property
    def qtdCandidatos(self):
        return self._candidatos.__len__()

    # Retorna uma representação string dos candidatos
    @property
    def strCandidatos(self):
        s = ''
        for candidato in self._candidatos.values():
            s += '{}\n'.format(candidato)
        return s

    # Retorna o dicionário de apartamentos
    @property
    def apartamentos(self):
        return self._apartamentos

    @property
    def qtdApartamentos(self):
        return self._apartamentos.__len__()

    # Retorna uam representação string dos apartamentos da forma de lista [1, 2, ...]
    # São listados 10 apartamentos por linha
    @property
    def strApartamentos(self):
        s = ''
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        for tupla in aptos:
            s += '{}'.format(tupla[1])
        return s

    # Retorna um objeto Apartamento através de seu número
    def apartamento(self, num_apto):
        return self._apartamentos[num_apto]

    # Registra um morador
    def registrarMorador(self, morador):
        # Confere tipo de objeto
        if not isinstance(morador, Morador):
            raise ErroMoradores('Parâmetro "morador" deve ser do tipo Morador ou Candidato')

        # Trata Apartamento
        if self._apartamentos.__contains__(morador.apartamento.numero):
            # Se o apartamento já havia sido cadastrado, recupera o Objeto para adicionar novo morador
            apto = self._apartamentos[morador.apartamento.numero]
        else:
            # Primeira ocorrência desse apartamento: adiciona-o ao dicionário
            apto = morador.apartamento
            self._apartamentos[apto.numero] = apto

        # Tenta adicionar morador ao apartamento, o que pode falhar se já existir outro morador com mesmo nome
```

```
        if apto.adicionarMorador(morador):
            # Não houve falha: Atualiza ligação entre Apartamento e Morador
            morador.apartamento = apto
        else:
            # Morador com mesmo nome já existe no Apartamento
            raise ErroMoradores('Morador duplicado')

        # Registra morador no dicionário
        self._moradores[morador.__hash__()] = morador

        # Se for um candidato, registra-o no dicionário específico
        if isinstance(morador, Candidato):
            self._candidatos[morador.__hash__()] = morador

    # Exclusão de um morador
    def excluirMorador(self, morador):
        # Confere o tipo de objeto
        if not isinstance(morador, Morador):
            raise ErroMoradores('Parâmetro "morador" deve ser do tipo Morador ou Candidato')

        # Remove o morador do Apartamento
        morador.apartamento.excluirMorador(morador)
        # Remove o morador do dicionário
        del self._moradores[morador.__hash__()]
        # Se o Apartamento não possui mais moradores, remove-o do dicionário de apartamentos
        if morador.apartamento.qtdMoradores == 0:
            del self._apartamentos[morador.apartamento.numero]
        # Se o morador excluído era um candidato, atualiza o dicionário de candidatos
        if isinstance(morador, Candidato):
            del self._candidatos[morador.__hash__()]

    # Permite carregar moradores, candidatos e apartamentos a partir de um arquivo CSV
    # O CSV deve ser delimitado por ponto-e-vírgula e deve conter três colunas, assim denominadas:
    #   Morador
    #   Apto
    #   Candidato?
    #
    # A coluna 'Morador' tem que conter um nome válido (ao menos um caractere)
    # A coluna 'Apto' deve conter um número inteiro maior que 0
    # A coluna 'Candidato?' deve conter N ou S  (Não ou Sim)
    #
    def carregarCSV(self, nomeArquivo):
        try:
            with open(nomeArquivo, 'r', newline='') as arqCSV:
                self._limpar()
                dados = csv.DictReader(arqCSV, delimiter=';')
                for reg in dados:
                    # Trata coluna 'Morador'
                    try:
                        nome = reg['Morador']
                    except KeyError:
                        raise ErroMoradores('Coluna "Morador" não localizada no arquivo CSV.')
```

```
                    # Traca coluna 'Apto'
                    try:
                        numApto = int(reg['Apto'])
                        try:
                            apto = self._apartamentos[numApto]
                        except:
                            apto = Apartamento(numApto)
                    except Exception:
                        raise ErroMoradores('Coluna "Apto" não localizada no arquivo CSV ou não contém um valor válido.')

                    # Trata coluna 'Candidato?'
                    try:
                        ehCandidato = reg['Candidato?'].upper() == 'S'
                    except:
                        raise ErroMoradores('Coluna "Candidato?" não localizada no arquivo CSV.')

                    # Instancia segundo o tipo
                    if ehCandidato:
                        morador = Candidato(nome, apto)
                    else:
                        morador = Morador(nome, apto)

                    # Registra o novo morador nos dicionários pertinentes
                    self.registrarMorador(morador)
        except:
            raise ErroMoradores('Falha ao abrir o arquivo CSV para leitura.')

    # Exibição da lista de candidatos
    def exibirCandidatos(self):
        print('Candidatos - quantidade = {}'.format(self.qtdCandidatos))
        print(self.strCandidatos)

    # Exibição da lista de apartamentos (apenas o número do apartamento)
    def exibir_apartamentos(self):
        print('Apartamentos - quantidade = {}'.format(self.qtdApartamentos))
        print(self.strApartamentos)

    # Representação visual do objeto
    def __repr__(self):
        # Relação de candidatos
        return 'CADASTRO\n' + \
               '========\n' + \
               'Candidatos:\n' + self.strCandidatos + \
               'Moradores:\n' + self.strMoradores + \
               'Apartamentos:\n' + self.strApartamentos


# ------------------------------------------------------------------------
# Classe Urna
#
# Representa uma urna de votação, com candidatos e registros de votos
```

```python
#
class Urna:

    # Construtor
    def __init__(self):
        self._limpar()

    # Limpeza para uma nova carga de dados
    def _limpar(self):
        self._apartamentos = {}
        self._votaram = 0
        self._candidatos = {}
        self._proxNumCandidato = 1

    # Registra um morador na urna (que pode ser um candidato ou não).
    def registrarMorador(self, morador):
        # Valida tipo de objeto passado como parâmetro
        if not isinstance(morador, Morador):
            raise ErroUrna('Parâmetro "morador" deve ser do tipo Morador')

        if isinstance(morador, Candidato):
            self.registrarCandidato(morador)    # Isso já registra o apartamento do candidato também
        else:
            self.registrarApartamento(morador.apartamento)

    # Registra um candidato na urna.
    def registrarCandidato(self, candidato):
        # Valida tipo de objeto passado como parâmetro
        if not isinstance(candidato, Candidato):
            raise ErroUrna('Parâmetro "candidato" deve ser do tipo Candidato')

        # Impede cadastramento em duplicidade
        if self._candidatos.__contains__(candidato):
            raise ErroUrna('Candidato duplicado')

        # Define número de candidato, registra e atualiza para novos candidatos futuros
        candidato.numero = self._proxNumCandidato
        self._candidatos[candidato.numero] = (candidato, 0)
        self._proxNumCandidato += 1
        # Registra o apartamento do candidato automaticamente
        self.registrarApartamento(candidato.apartamento)

    # Registra apartamento para um morador votante.
    def registrarApartamento(self, apartamento) -> None:
        self._apartamentos[apartamento.numero] = apartamento

    # Retorna a quantidade de candidatos
    @property
    def qtdCandidatos(self):
        return self._candidatos.__len__()

    # Retorna uma representação string dos candidatos
```

```
    @property
    def strCandidatos(self):
        s = 'Candidatos: {}\n'.format(self.qtdCandidatos)
        for tupla in self._candidatos.values():
            candidato = tupla[0]
            s += '{} - {} (Apto {})\n'.format(candidato.numero, candidato.nome, candidato.apartamento.numero)
        return s

    # Exibição da lista de candidatos
    def exibirCandidatos(self):
        print(self.strCandidatos)

    # Retorna a quantidade de apartamentos
    @property
    def qtdApartamentos(self):
        return self._apartamentos.__len__()

    # Retorna uam representação string dos apartamentos
    @property
    def strApartamentos(self):
        s = 'Apartamentos: {}\n['.format(self.qtdApartamentos)
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        sep = ''
        for apto in aptos:
            s += sep + apto.numero()
            sep = ','
        s += ']\n'
        return s

    # Retorna um objeto Apartamento por seu número
    def apartamento(self, num):
        try:
            apto = self._apartamentos[num]
        except:
            raise ErroUrna('Apartamento {} não localizado'.format(num))
        return apto

    # Retorna True ou False se o apartamento já votou
    def apartamentoVotou(self, num):
        return self.apartamento(num).votou

    # Retorna True se o número informado constar na relação de candidatos registrados na Urna
    def numCandidatoValido(self, num):
        return self._candidatos.__contains__(num)

    # votantes = total de apartamentos votantes (apenas um sinônimo para qtdApartamentos nessa versão)
    @property
    def votantes(self):
        return self.qtdApartamentos

    # Retorna True se a votação foi iniciada
    @property
```

```
    def votacaoIniciada(self):
        return self._votaram > 0

    # Retorna True se a votação está terminada
    @property
    def votacaoTerminada(self):
        return self.votantes == self._votaram

    # Registra voto de um apartamento em um candidato
    def registrarVoto(self, apartamento, numCandidato):
        # Votação encerrada não admite novos votos
        if self.votacaoTerminada:
            raise ErroUrna('A votação já está terminada. Voto ignorado.')

        # Um apartamento não pode votar duas vezes
        if self.apartamentoVotou(apartamento.numero):
            raise ErroUrna('Apartamento já votou.')

        # Obtem o objeto Candidato e os votos contabilizados para ele
        try:
            tupla = self._candidatos[numCandidato]
            candidato = tupla[0]
            votos = tupla[1]
        except:
            raise ErroUrna('Candidato {} não encontrado.'.format(numCandidato))

        # Atualiza a contagem de votos do objeto Candidato
        # Atualiza a contagem local de votos para o Candidato (duplo check - vide método auditar()
        # Atualiza status de votação do apartamento
        # Atualiza a contagem geral de votos (que controla o término da votação)
        candidato.registrarVoto()
        self._candidatos[numCandidato] = (candidato, votos + 1)
        apartamento.votou = True
        self._votaram += 1

    # Retorna uma lista contendo:
    # 0: representação string do status da votação atual
    # 1: lista de nomes dos candidatos
    # 2: total de votos dos candidatos
    #
    # Viabiliza a apresentação textual dos resultados e a formatação de um gráfico correspondente
    #
    @property
    def resultados(self):
        candidatos = []
        votos = []
        # Votação em andamento? Se ainda não iniciou não há resultado algum a apresentar
        if self._votaram > 0:
            # Audita os resultados antes de apresentar
            if self.votacaoTerminada:
                self.auditar()
                s = '\nVotação terminada.\n'
```

```
            else:
                s = '\nVotação em andamento. Quantidades PARCIAIS\n'
            # Carrega candidatos e votos numa lista para viabilizar a ordenação por quantidade decrescente de votos
            totais = []
            for tupla in self._candidatos.values():
                candidato = tupla[0]
                qtd_votos = tupla[1]
                tupla = (candidato.numero, candidato.nome, qtd_votos)
                totais.append(tupla)

            # Ordena e apresenta os resultados
            totais.sort(key=lambda item: item[2], reverse=True)
            s += '\nRESULTADOS:\n'
            for item in totais:
                s += '{} - {} = {} votos\n'.format(item[0], item[1], item[2])
                candidatos.append(item[1])
                votos.append(item[2])
        else:
            s = '\nVotação não iniciada.\n'
        return [s, candidatos, votos]

    # Audita o resultado, conferindo se a soma de votos da urna é igual a soma de votos dos objetos Candidato
    # individuais
    def auditar(self):
        # Valida, para cada Candidato, o total de votos versus o total que consta na urna
        somaVotos = 0
        for tupla in self._candidatos.values():
            # tupla[0] = total de votos no objeto Candidato
            # tupla[1] = total de votos na urna
            if tupla[0].votos != tupla[1]:
                raise ErroUrna('Tentativa de fraudar votação. Candidato {}. Votos válidos={}, Votos do candidato={}.'.
                               format(tupla[0], tupla[1], tupla[0].votos))
            somaVotos += tupla[1]

        # O total de votos deve ser igual ao total de votantes ou outra forma de adulterar a urna foi empregada
        if somaVotos != self._votaram:
            raise ErroUrna('Erro inesperado. Total de votos não confere. Registrados={}, Contabilizados={}'.format(
                self._votaram, somaVotos))

    # Exibição dos resultados
    def exibirResultados(self):
        # Obtem os dados atualizados da urna
        dados = self.resultados
        # Configura o título
        if self.votacaoTerminada:
            titulo = 'Resultado final das eleições- {} votos computados'.format(self._votaram)
        else:
            titulo = 'Resultado PARCIAL das eleições - {} votos computados'.format(self._votaram)
        # Apresenta gráfico de barras
        plt.bar(dados[1], dados[2])
        plt.title(titulo)
        plt.xlabel('Candidatos')
```

```
        plt.ylabel('Votos')
        plt.show()
        # Apresenta dados textuais
        print(dados[0])
        print()

    # Representação visual do objeto
    def __repr__(self):
        # Relação de candidatos
        s = 'URNA\n' + self.strCandidatos
        # Relação de apartamentos votantes
        s += self.strApartamentos
        # Total de votos computados
        s += 'Total de votos computados: {}\n\n'.format(self._votaram)
        # Resultados da votação
        s += self.resultados
        return s


# ------------------------------------------------------------------------
# Classe Menu
#
# Execução das tarefas de suporte para votação
#
# Atributos:
# - Dicionário de moradores, candidatos e apartamentos
# - Objeto Urna para armazenamento dos votos
#
class Menu:

    # Construtor
    def __init__(self):
        self._moradores = Moradores()   # Dicionário vazio de moradores
        self._urna = None               # Urna não preparada para votação

    # Método utilitário para apresentar títulos em tela para interação com usuário
    def titulo(self, msg):
        qtd_carac = len(msg)
        print()
        print("=" * qtd_carac)
        print(msg)
        print("=" * qtd_carac)

    # Método utilitário para apresentar uma mensagem e aguardar pela tecla Enter
    def _tecleEnter(self, msg=''):
        msg += ' <Tecle Enter para continuar>'
        input(msg)

    # Método utilitário para apresentar uma mensagem e arguardar confirmação do usuário
    def _Confirmar(self, msg=''):
        opcao = ' '
        while opcao not in ['S','N']:
```

```
            opcao = input(msg + ' (S/N)? ').upper()
        return opcao == 'S'

    # Laço do menu principal
    def executar(self):
        opcao = 1
        while opcao > 0:
            self.titulo('MENU PRINCIPAL')
            print('1 - Cadastro de moradores e candidatos')
            print('2 - Configurar urna de votação')
            print('3 - Descartar urna configurada')
            print('4 - Votação')
            print('5 - Resultados')
            print('0 - Encerrar')
            try:
                opcao = int(input('Opção: '))
                if opcao == 1:
                    # Cadastramentos
                    self._menuCadastro()
                elif opcao == 2:
                    # Configurar urna
                    self._configurarUrna()
                elif opcao == 3:
                    # Descartar urna configurada
                    self._descartarUrna()
                elif opcao == 4:
                    # Votação
                    if self._urna is None:
                        self._tecleEnter('Urna ainda não configurada. Votação não iniciada.')
                    else:
                        try:
                            self.__votar()
                        except Exception as e:
                            self._tecleEnter(e.__str__())
                elif opcao == 5:
                    # Resultados até o momento...
                    if self._urna is None:
                        self._tecleEnter('Votação ainda não iniciada.')
                    else:
                        self.titulo('RESULTADOS')
                        try:
                            self._urna.exibirResultados()
                        except Exception as e:
                            print(e.__str__())
                            opcao = 1       # Para continuar no menu
                        self._tecleEnter()
                elif opcao == 0:
                    # Finalizar execução
                    if self._Confirmar('Confirma encerramento'):
                        break
                    else:
                        opcao = 1    # Para continuar no menu
```

```
                else:
                    raise ErroMenu()
            except Exception:
                print('Opção inválida.')

    # Cadastramento de moradores, candidatos e apartamentos
    def _menuCadastro(self):
        opcao = 1
        while opcao > 0:
            self.titulo('MANUTENÇÃO CADASTRAL')
            print('1 - Cadastrar candidato ou morador')
            print('2 - Alterar ou excluir morador ou candidato')
            print('3 - Carregar em lote a partir de arquivo CSV')
            print('4 - Listar candidatos')
            print('5 - Listar apartamentos e moradores')
            print('0 - Voltar')
            try:
                opcao = int(input('Opção: '))

                if opcao > 0 and opcao < 4 and self._urna is not None:
                    # Operações não autorizadas com votação em andamento
                    self._tecleEnter('Não autorizado: Votação iniciada.')
                elif opcao == 1:
                    # Cadastrar morador ou candidato
                    self._cadastrarCandidatoMorador()
                elif opcao == 2:
                    # Alterar ou excluir morador ou candidato
                    self._editarCandidatoMorador()
                elif opcao == 3:
                    # Carregar dados de arquio CSV
                    self._carregarCSV()
                elif opcao == 4:
                    # Listar candidatos
                    self.titulo('LISTA DE CANDIDATOS')
                    self._moradores.exibirCandidatos()
                    self._tecleEnter()
                elif opcao == 5:
                    # Listar apartamentos e moradores
                    self.titulo('LISTA DE APARTAMENTOS E MORADORES')
                    self._moradores.exibir_apartamentos()
                    self._tecleEnter()
                elif opcao == 0:
                    # Voltar ao menu anterior
                    return
                else:
                    # Valor de opção inválida
                    raise ErroMenu()
            except Exception as e:
                print('Opção inválida. '+ e.__str__())

    # Cadastramento de candidato ou morador
    def _cadastrarCandidatoMorador(self):
```

```
        self.titulo("CADASTRAMENTO DE CANDIDATO/MORADOR")
        # Pergunta o tipo de pessoa a cadastrar
        tipo = ' '
        while tipo not in ['C','M','V']:
            tipo = input("Candidato, Morador ou Voltar (C/M/V)? ").upper()

        if tipo == 'V':
            # Retornar ao menu
            return

        # Instancia de acordo com o tipo
        if tipo == 'C':
            morador = Candidato()
        else:
            morador = Morador()

        # Registra morador ou candidato criado
        try:
            self._moradores.registrarMorador(morador)
            msg = 'Cadastramento bem sucedido.'
        except Exception as e:
            msg = e.__str__()
        self._tecleEnter(msg)

    # Alteração ou Exclusão de Candidato ou Morador
    def _editarCandidatoMorador(self):
        apto = 1
        while apto is not None:
            self.titulo('EDIÇÃO DE CANDIDATO/MORADOR')
            # Primeiro passo: Seleção do apartamento
            apto = self._selecaoApto()    # Retorna None se o usuário cancelar a seleção
            morador = 1
            while apto is not None and morador is not None:
                # Segundo passo é a seleção do morador.
                morador = self._selecaoMorador(apto)
                if morador is not None:       # Usuário cancelou a operação - volta para seleção de apartamento
                    # Terceiro passo é a seleção da operação
                    acao = input('Deseja (A)lterar ou (E)xcluir (qualquer outro valor para cancelar)? ').upper()
                    if acao == 'A':
                        # Alterar nome do morador
                        self._alterarCandidatoMorador(morador)
                    elif acao == 'E':
                        # Excluir morador
                        self._excluirCandidatoMorador(apto, morador)
                        if apto.qtdMoradores == 0:
                            self._tecleEnter('Nenhum morador mais no apartamento. Apartamento excluído.')
                            morador = None

    # Alteração do nome do candidato/morador
    def _alterarCandidatoMorador(self, morador):
        self.titulo('ALTERAÇÃO DE CANDIDATO/MORADOR')
        print('Nome atual:', morador.nome)
```

```
        nomeValido = False
        while not nomeValido:
            novoNome = input('Novo nome.: ')
            nomeValido = len(novoNome.strip(' ')) > 0
            if nomeValido and self._Confirmar('Confirma a alteração de nome?'):
                morador.nome = novoNome
                self._tecleEnter('Alteração concluida.')
            else:
                nomeValido = True

    # Exclusão do candidato/morador
    def _excluirCandidatoMorador(self, apartamento, morador):
        self.titulo('EXCLUSÃO DE CANDIDATO/MORADOR')
        if self._Confirmar('Confirma a exclusão do morador "{}"'.format(morador.nome)):
            self._moradores.excluirMorador(morador)
            self._tecleEnter('Exclusão concluida.')

    # Carregar apartamentos, moradores e candidatos a partir de um arquivo CSV
    def _carregarCSV(self):
        self.titulo("CARGA DE ARQUIVO CSV")
        arquivo = input('Caminho e nome do arquivo CSV: ')
        try:
            self._moradores.carregarCSV(arquivo)
            msg = 'Carregamento bem sucedido.'
        except Exception as e:
            msg = e.__str__()
        self._tecleEnter(msg)

    # Configuração da urna para início de votação
    def _configurarUrna(self):
        self.titulo('CONFIGURAÇÃO DA URNA DE VOTAÇÃO')

        # Pelo menos 2 candidatos
        if self._moradores.qtdCandidatos < 2:
            self._tecleEnter('Para iniciar votação são necessários 2 candidatos ou mais.')
            return

        # Pelo menos 2 apartamentos
        if self._moradores.qtdApartamentos < 2:
            self._tecleEnter('Para iniciar votação são necessários 2 apartamentos ou mais.')
            return

        # Registra os candidatos na urna
        if self._urna is None:
            self._urna = Urna()
            for candidato in self._moradores.candidatos.values():
                self._urna.registrarCandidato(candidato)

            # Registra os apartamentos votantes na urna
            for apto in self._moradores.apartamentos.values():
                self._urna.registrarApartamento(apto)
            self._tecleEnter('Configuração bem sucedida.')
```

```
        else:
            self._tecleEnter('Atenção: Urna já estava configurada.')

    # Descarte de urna configurada para votação
    def _descartarUrna(self):
        self.titulo('DESCARTE DE URNA')
        if self._urna is None:
            msg = 'Nenhuma urna ativa. Nada a fazer.'

        elif self._urna.votacaoIniciada:
            msg = 'Votação em andamento. Urna não pode ser descartada.'

        elif self._Confirmar('Confirma descarte da urna'):
            self._urna = None
            msg = 'Urna reinicializada.'

        else:
            msg = 'Descarte cancelado. Urna continua ativa para início de votação.'
        self._tecleEnter(msg)

    # Votação
    def __votar(self):
        # Executa loop até todos que todos os apartamentos tenham votado
        while not self._urna.votacaoTerminada:
            self.titulo('VOTAÇÃO')
            # Antes do primeiro voto alerta que a urna não pode mais ser alterada
            if not (self._urna.votacaoIniciada or
                    self._Confirmar('Se a votação for iniciada não poderá mais ser interrompida. Confirma')):
                return

            # Primeiro passo da votação: Seleção do apartamento do votante
            apto = self._selecaoApto()
            if apto is None:
                # Sair do loop de votação e voltar ao menu principal
                # Votação pode ser retomada normalmente depois
                break

            # Não permite que um apartamento vote mais que uma vez
            if self._urna.apartamentoVotou(apto.numero):
                self._tecleEnter('Esse apartamento já votou.')
                continue

            # ALTERNATIVA: Utilizar o primeiro objeto Morador do objeto Apartamento, pois como testamos previamente
            # contra duplicidade de votação e saber quem está votando do apartamento não é relevante.
            apto.morador(0).votar(self._urna)

            # ALTERNATIVA: Exibir os moradores e permitir que o votante se identifique
            #
            # Segundo passo é a seleção do morador.
            # Se retornar None o usuário selecionou "cancelar" e o program deve voltar para a seleção de apartamento
            # do votante
            #morador = self._selecaoMorador(apto)
```

```
            #if morador is not None:
            #    # Terceiro passo é a seleção do candidato (Voto)
            #    morador.votar(self._urna)

            # Aguarda alguns segundos antes de preparar para o próximo voto, para o morador visualizar a confirmação
            # de voto registrado
            if self._urna.votacaoTerminada:
                print('* * * Votação Encerrada * * *')
            else:
                print('Preparando para próximo voto...')
            time.sleep(2)

    # Seleção do apartamento onde mora a pessoa que irá votar
    def _selecaoApto(self):
        apto = None
        while apto is None:
            numApto = input('Informe o número do apartamento (0 para cancelar): ')
            try:
                numApto = int(numApto)
                # Usuário deseja cancelar e voltar
                if numApto == 0:
                    break
                # Seleciona o apartamento
                apto = self._moradores.apartamento(numApto)
            except:
                print('Apartamento inválido!')
        return apto

    # Seleção do obter_morador que irá votar
    def _selecaoMorador(self, apto):
        morador = None
        while morador is None:
            apto.exibirMoradores()
            try:
                num = int(input('Selecione morador (0 para cancelar): '))
                if num < 0 or num > apto.qtdMoradores:
                    raise Exception()
                # Usuário deseja cancelar e voltar
                if num == 0:
                    return None
                # Seleciona o objeto Morador correspondente ao número escolhido
                morador = apto.morador(num-1)
            except:
                self._tecleEnter('Número de morador inválido.')
        return morador


# -----------------------------------------------------------------------------
# Programa Principal
#
menu = Menu()
menu.executar()
```

```
==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar
```

In [2]:

```python
# ------------------------------------------------------------------------------
# Let's Code
# Projeto 2 - Sistema para votação de síndico
# por Alexandre Rozante
#
import csv
import time
import matplotlib.pyplot as plt

# ------------------------------------------------------------------------------
# Exceções
class ErroApartamento(Exception):
    pass

class ErroMorador(Exception):
    pass

class ErroCandidato(Exception):
    pass

class ErroMoradores(Exception):
    pass

class ErroUrna(Exception):
    pass

class ErroMenu(Exception):
    pass

# ------------------------------------------------------------------------------
# Classe Apartamento
#
# Representa um apartamento contendo moradores
#
# Atributos:
# - Número do apartamento
# - Lista de moradores
# - Flag de status de votação (True ou False)
#
class Apartamento:
```

```
    # Construtor
    def __init__(self, numero):
        if type(numero) != int:
            raise ErroApartamento('Um parâmetro tipo int é esperado.')
        self._numero = numero      # Número do apartamento
        self._moradores = []       # Lista de moradores
        self._votou = False        # Status de votação

    # Número do apartamento
    @property
    def numero(self):
        return self._numero

    # Quantidade de moradores
    @property
    def qtdMoradores(self):
        return self._moradores.__len__()

    # Exibe os moradores
    @property
    def strMoradores(self):
        s = 'Moradores:\n'
        for i in range(self.qtdMoradores):
            s += '   {}) {}\n'.format((i + 1), self._moradores[i].nome)
        return s

    # Retorna um morador pelo seu índice na lista
    def morador(self, indice):
        return self._moradores[indice]

    # Status de votação
    @property
    def votou(self):
        return self._votou

    @votou.setter
    def votou(self, votou):
        if type(votou) != bool:
            raise ErroApartamento('Um parâmetro bool é esperado.')
        self._votou = votou

    # Adição de Morador ao Apartamento
    def adicionarMorador(self, morador):
        nomeNovoMorador = morador.nome.lower().strip(' ')
        for moradorExistente in self._moradores:
            if moradorExistente.nome.lower().strip(' ') == nomeNovoMorador:
                return False
        self._moradores.append(morador)
        return True

    # Exclusão de Morador
    def excluirMorador(self, morador):
```

```
        self._moradores.remove(morador)

    # Exibe os moradores
    def exibirMoradores(self):
        print(self.strMoradores)

    # Representação visual do objeto
    def __repr__(self):
        return 'Apto={}\n'.format(self.numero) + self.strMoradores


# ------------------------------------------------------------------------
# Classe Morador
#
# Representa um morador de um apartamento
#
# Atributos:
# - Nome do obter_morador
# - Apartamento em que reside (objeto)
#
# Caso o nome não seja informado o método solicita sua digitação.
#
# Caso o apartamento não seja informado o método solicita seu número e instancia
# um novo objeto Apartamento.
#
class Morador:

    # Construtor
    def __init__(self, nome='', apto=None):
        # Apesar do nome ser opcional, Python permite que valores de diferentes tipos sejam passados.
        # Verificamos se é um valor String, e caso negativo, forçamos digitação ignorando o valor informado.
        if type(nome) != str:
            raise ErroMorador('Tipo inválido para o parâmetro "nome" do objeto Morador')

        # Procedimento similar quanto ao tipo de dado do parâmetro apto (Apartamento)
        if apto is not None and type(apto) != Apartamento:
            raise ErroMorador('Tipo inválido para o parâmetro "apto" do objeto Morador')

        # Nome não especificado? Solicita digitação
        while len(nome.strip(' ')) < 1:
            # Ao menos um caractere para o nome do morador deve ser digitado
            nome = input('Favor digitar o nome....: ')

        # Apartamento não especificado?
        if apto is None:
            # Solicita um número de apartamento maior que zero
            num_apto = 0
            while num_apto < 1:
                try:
                    num_apto = int(input('Número do apartamento...: '))
                except ValueError:
                    print('Número inválido.')
```

```
            # Instancia um novo apartamento para o obter_morador
            apto = Apartamento(num_apto)

        self._nome = nome
        self._apartamento = apto

    # Nome do morador
    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, novo_nome):
        self._nome = novo_nome

    # Apartamento
    @property
    def apartamento(self):
        return self._apartamento

    @apartamento.setter
    def apartamento(self, apto):
        self._apartamento = apto

    # Registro de voto do morador
    def votar(self, urna, numCandidato=0):
        interativo = False

        # Se não for especificada uma urna valida aborta o método
        if not isinstance(urna, Urna):
            raise ErroMorador('Objeto urna inválido')

        # Se um valor não numérico ou não inteiro for passado como número de candidato, força digitação
        if type(numCandidato) != int:
            raise ErroMorador('Número do candidato deve ser int.')

        # Se o número de candidato informado não existe, força digitação
        if numCandidato > 0:
            if not urna.numCandidatoValido(numCandidato):
                raise ErroMorador('Candidato inexistente {}'.format(numCandidato))
        else:
            # Enquanto um número válido de candidato não for especificado continua solicitando
            while numCandidato < 1:
                interativo = True
                # Apresenta a relação de candidatos e aguarda votação
                urna.exibirCandidatos()
                try:
                    numCandidato = int(input('Deseja votar em qual candidato? '))
                    if not urna.numCandidatoValido(numCandidato):
                        numCandidato = 0
                        raise ErroMorador()
                except Exception:
```

```
                    print('Candidato inválido')
        # Registra o voto realizado
        urna.registrarVoto(self.apartamento, numCandidato)
        if interativo:
            print('\n* * * Voto registrado com sucesso * * *.\n')

    # Representação visual do objeto
    def __repr__(self):
        return 'Morador {}, Apto.{}'.format(self.nome, self.apartamento.numero)


# -----------------------------------------------------------------------------
# Classe Candidato (extende Morador)
#
# Representa um morador que é um candidato a síndico
#
# Atributos:
# - Nome do obter_morador
# - Apartamento
# - Número de candidato
# - Votos recebidos
#
class Candidato(Morador):

    # Construtor
    def __init__(self, nome='', apto=None):
        super().__init__(nome, apto)
        self._numero = 0
        self._votos = 0

    # Número de candidato (r/w)
    @property
    def numero(self):
        return self._numero

    @numero.setter
    def numero(self, num):
        if type(num) != int:
            raise ErroCandidato('Parâmetro num deve ser um int')
        if num < 1:
            raise ErroCandidato('Parâmetro num deve ser um valor positivo maior que 0')
        self._numero = num

    # Votos recebidos é somente leitura.
    # Um método específico para incrementar unitariamente foi criado para maior segurança do processo de votação
    @property
    def votos(self):
        return self._votos

    # Registra um voto para o candidato
    def registrarVoto(self):
        self._votos += 1
```

```python
    # Representação visual do objeto
    def __repr__(self):
        reprMorador = super().__repr__()
        return 'Candidato {} - {} - Votos: {}'.format(self._numero, reprMorador, self._votos)


# -----------------------------------------------------------------------------
# Classe Moradores
#
# Armazena apartamentos, moradores e candidatos para um processo de eleição de
# síndico
#
# Atributos:
# - Dicionário de moradores (que podem ser candidatos ou não)
# - Dicionário de candidatos
# - Dicionário de apartamentos
#
class Moradores:

    # Construtor
    def __init__(self):
        self._limpar()

    # Limpeza para uma nova carga de dados
    # É utilizado no instanciamento de novos objetos e pelo método carregarCSV
    def _limpar(self):
        self._moradores = {}
        self._candidatos = {}
        self._apartamentos = {}

    # Retorna a quantidade de moradores cadastrados
    @property
    def qtdMoradores(self):
        return self._moradores.__len__()

    # Retorna uma representação string dos moradores
    @property
    def strMoradores(self):
        # Como podemos ter moradores homônimos em apartamentos distintos, logo informamos o apartamento de cada morador
        # É conveniente exibir em ordem crescente de nº de apartamento
        s = ''
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        for apto in aptos:
            s += '{}\n'.format(apto[1])
        return s

    # Retorna o dicionário de candidatos
    @property
    def candidatos(self):
        return self._candidatos
```

```python
    # Retorna a quantidade de candidatos cadastrados
    @property
    def qtdCandidatos(self):
        return self._candidatos.__len__()

    # Retorna uma representação string dos candidatos
    @property
    def strCandidatos(self):
        s = ''
        for candidato in self._candidatos.values():
            s += '{}\n'.format(candidato)
        return s

    # Retorna o dicionário de apartamentos
    @property
    def apartamentos(self):
        return self._apartamentos

    @property
    def qtdApartamentos(self):
        return self._apartamentos.__len__()

    # Retorna uam representação string dos apartamentos da forma de lista [1, 2, ...]
    # São listados 10 apartamentos por linha
    @property
    def strApartamentos(self):
        s = ''
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        for tupla in aptos:
            s += '{}'.format(tupla[1])
        return s

    # Retorna um objeto Apartamento através de seu número
    def apartamento(self, num_apto):
        return self._apartamentos[num_apto]

    # Registra um morador
    def registrarMorador(self, morador):
        # Confere tipo de objeto
        if not isinstance(morador, Morador):
            raise ErroMoradores('Parâmetro "morador" deve ser do tipo Morador ou Candidato')

        # Trata Apartamento
        if self._apartamentos.__contains__(morador.apartamento.numero):
            # Se o apartamento já havia sido cadastrado, recupera o Objeto para adicionar novo morador
            apto = self._apartamentos[morador.apartamento.numero]
        else:
            # Primeira ocorrência desse apartamento: adiciona-o ao dicionário
            apto = morador.apartamento
            self._apartamentos[apto.numero] = apto

        # Tenta adicionar morador ao apartamento, o que pode falhar se já existir outro morador com mesmo nome
```

```
        if apto.adicionarMorador(morador):
            # Não houve falha: Atualiza ligação entre Apartamento e Morador
            morador.apartamento = apto
        else:
            # Morador com mesmo nome já existe no Apartamento
            raise ErroMoradores('Morador duplicado')

        # Registra morador no dicionário
        self._moradores[morador.__hash__()] = morador

        # Se for um candidato, registra-o no dicionário específico
        if isinstance(morador, Candidato):
            self._candidatos[morador.__hash__()] = morador

    # Exclusão de um morador
    def excluirMorador(self, morador):
        # Confere o tipo de objeto
        if not isinstance(morador, Morador):
            raise ErroMoradores('Parâmetro "morador" deve ser do tipo Morador ou Candidato')

        # Remove o morador do Apartamento
        morador.apartamento.excluirMorador(morador)
        # Remove o morador do dicionário
        del self._moradores[morador.__hash__()]
        # Se o Apartamento não possui mais moradores, remove-o do dicionário de apartamentos
        if morador.apartamento.qtdMoradores == 0:
            del self._apartamentos[morador.apartamento.numero]
        # Se o morador excluído era um candidato, atualiza o dicionário de candidatos
        if isinstance(morador, Candidato):
            del self._candidatos[morador.__hash__()]

    # Permite carregar moradores, candidatos e apartamentos a partir de um arquivo CSV
    # O CSV deve ser delimitado por ponto-e-vírgula e deve conter três colunas, assim denominadas:
    #   Morador
    #   Apto
    #   Candidato?
    #
    # A coluna 'Morador' tem que conter um nome válido (ao menos um caractere)
    # A coluna 'Apto' deve conter um número inteiro maior que 0
    # A coluna 'Candidato?' deve conter N ou S  (Não ou Sim)
    #
    def carregarCSV(self, nomeArquivo):
        try:
            with open(nomeArquivo, 'r', newline='') as arqCSV:
                self._limpar()
                dados = csv.DictReader(arqCSV, delimiter=';')
                for reg in dados:
                    # Trata coluna 'Morador'
                    try:
                        nome = reg['Morador']
                    except KeyError:
                        raise ErroMoradores('Coluna "Morador" não localizada no arquivo CSV.')
```

```
                    # Traca coluna 'Apto'
                    try:
                        numApto = int(reg['Apto'])
                        try:
                            apto = self._apartamentos[numApto]
                        except:
                            apto = Apartamento(numApto)
                    except Exception:
                        raise ErroMoradores('Coluna "Apto" não localizada no arquivo CSV ou não contém um valor válido.')

                    # Trata coluna 'Candidato?'
                    try:
                        ehCandidato = reg['Candidato?'].upper() == 'S'
                    except:
                        raise ErroMoradores('Coluna "Candidato?" não localizada no arquivo CSV.')

                    # Instancia segundo o tipo
                    if ehCandidato:
                        morador = Candidato(nome, apto)
                    else:
                        morador = Morador(nome, apto)

                    # Registra o novo morador nos dicionários pertinentes
                    self.registrarMorador(morador)
        except:
            raise ErroMoradores('Falha ao abrir o arquivo CSV para leitura.')

    # Exibição da lista de candidatos
    def exibirCandidatos(self):
        print('Candidatos - quantidade = {}'.format(self.qtdCandidatos))
        print(self.strCandidatos)

    # Exibição da lista de apartamentos (apenas o número do apartamento)
    def exibir_apartamentos(self):
        print('Apartamentos - quantidade = {}'.format(self.qtdApartamentos))
        print(self.strApartamentos)

    # Representação visual do objeto
    def __repr__(self):
        # Relação de candidatos
        return 'CADASTRO\n' + \
               '========\n' + \
               'Candidatos:\n' + self.strCandidatos + \
               'Moradores:\n' + self.strMoradores + \
               'Apartamentos:\n' + self.strApartamentos


# ------------------------------------------------------------------------
# Classe Urna
#
# Representa uma urna de votação, com candidatos e registros de votos
```

```python
#
class Urna:

    # Construtor
    def __init__(self):
        self._limpar()

    # Limpeza para uma nova carga de dados
    def _limpar(self):
        self._apartamentos = {}
        self._votaram = 0
        self._candidatos = {}
        self._proxNumCandidato = 1

    # Registra um morador na urna (que pode ser um candidato ou não).
    def registrarMorador(self, morador):
        # Valida tipo de objeto passado como parâmetro
        if not isinstance(morador, Morador):
            raise ErroUrna('Parâmetro "morador" deve ser do tipo Morador')

        if isinstance(morador, Candidato):
            self.registrarCandidato(morador)    # Isso já registra o apartamento do candidato também
        else:
            self.registrarApartamento(morador.apartamento)

    # Registra um candidato na urna.
    def registrarCandidato(self, candidato):
        # Valida tipo de objeto passado como parâmetro
        if not isinstance(candidato, Candidato):
            raise ErroUrna('Parâmetro "candidato" deve ser do tipo Candidato')

        # Impede cadastramento em duplicidade
        if self._candidatos.__contains__(candidato):
            raise ErroUrna('Candidato duplicado')

        # Define número de candidato, registra e atualiza para novos candidatos futuros
        candidato.numero = self._proxNumCandidato
        self._candidatos[candidato.numero] = (candidato, 0)
        self._proxNumCandidato += 1
        # Registra o apartamento do candidato automaticamente
        self.registrarApartamento(candidato.apartamento)

    # Registra apartamento para um morador votante.
    def registrarApartamento(self, apartamento) -> None:
        self._apartamentos[apartamento.numero] = apartamento

    # Retorna a quantidade de candidatos
    @property
    def qtdCandidatos(self):
        return self._candidatos.__len__()

    # Retorna uma representação string dos candidatos
```

```
    @property
    def strCandidatos(self):
        s = 'Candidatos: {}\n'.format(self.qtdCandidatos)
        for tupla in self._candidatos.values():
            candidato = tupla[0]
            s += '{} - {} (Apto {})\n'.format(candidato.numero, candidato.nome, candidato.apartamento.numero)
        return s

    # Exibição da lista de candidatos
    def exibirCandidatos(self):
        print(self.strCandidatos)

    # Retorna a quantidade de apartamentos
    @property
    def qtdApartamentos(self):
        return self._apartamentos.__len__()

    # Retorna uam representação string dos apartamentos
    @property
    def strApartamentos(self):
        s = 'Apartamentos: {}\n['.format(self.qtdApartamentos)
        aptos = sorted(self._apartamentos.items(), key=lambda item: item[0])
        sep = ''
        for apto in aptos:
            s += sep + apto.numero()
            sep = ','
        s += ']\n'
        return s

    # Retorna um objeto Apartamento por seu número
    def apartamento(self, num):
        try:
            apto = self._apartamentos[num]
        except:
            raise ErroUrna('Apartamento {} não localizado'.format(num))
        return apto

    # Retorna True ou False se o apartamento já votou
    def apartamentoVotou(self, num):
        return self.apartamento(num).votou

    # Retorna True se o número informado constar na relação de candidatos registrados na Urna
    def numCandidatoValido(self, num):
        return self._candidatos.__contains__(num)

    # votantes = total de apartamentos votantes (apenas um sinônimo para qtdApartamentos nessa versão)
    @property
    def votantes(self):
        return self.qtdApartamentos

    # Retorna True se a votação foi iniciada
    @property
```

```
    def votacaoIniciada(self):
        return self._votaram > 0

    # Retorna True se a votação está terminada
    @property
    def votacaoTerminada(self):
        return self.votantes == self._votaram

    # Registra voto de um apartamento em um candidato
    def registrarVoto(self, apartamento, numCandidato):
        # Votação encerrada não admite novos votos
        if self.votacaoTerminada:
            raise ErroUrna('A votação já está terminada. Voto ignorado.')

        # Um apartamento não pode votar duas vezes
        if self.apartamentoVotou(apartamento.numero):
            raise ErroUrna('Apartamento já votou.')

        # Obtem o objeto Candidato e os votos contabilizados para ele
        try:
            tupla = self._candidatos[numCandidato]
            candidato = tupla[0]
            votos = tupla[1]
        except:
            raise ErroUrna('Candidato {} não encontrado.'.format(numCandidato))

        # Atualiza a contagem de votos do objeto Candidato
        # Atualiza a contagem local de votos para o Candidato (duplo check - vide método auditar()
        # Atualiza status de votação do apartamento
        # Atualiza a contagem geral de votos (que controla o término da votação)
        candidato.registrarVoto()
        self._candidatos[numCandidato] = (candidato, votos + 1)
        apartamento.votou = True
        self._votaram += 1

    # Retorna uma lista contendo:
    # 0: representação string do status da votação atual
    # 1: lista de nomes dos candidatos
    # 2: total de votos dos candidatos
    #
    # Viabiliza a apresentação textual dos resultados e a formatação de um gráfico correspondente
    #
    @property
    def resultados(self):
        candidatos = []
        votos = []
        # Votação em andamento? Se ainda não iniciou não há resultado algum a apresentar
        if self._votaram > 0:
            # Audita os resultados antes de apresentar
            if self.votacaoTerminada:
                self.auditar()
                s = '\nVotação terminada.\n'
```

```
            else:
                s = '\nVotação em andamento. Quantidades PARCIAIS\n'
            # Carrega candidatos e votos numa lista para viabilizar a ordenação por quantidade decrescente de votos
            totais = []
            for tupla in self._candidatos.values():
                candidato = tupla[0]
                qtd_votos = tupla[1]
                tupla = (candidato.numero, candidato.nome, qtd_votos)
                totais.append(tupla)

            # Ordena e apresenta os resultados
            totais.sort(key=lambda item: item[2], reverse=True)
            s += '\nRESULTADOS:\n'
            for item in totais:
                s += '{} - {} = {} votos\n'.format(item[0], item[1], item[2])
                candidatos.append(item[1])
                votos.append(item[2])
        else:
            s = '\nVotação não iniciada.\n'
        return [s, candidatos, votos]

    # Audita o resultado, conferindo se a soma de votos da urna é igual a soma de votos dos objetos Candidato
    # individuais
    def auditar(self):
        # Valida, para cada Candidato, o total de votos versus o total que consta na urna
        somaVotos = 0
        for tupla in self._candidatos.values():
            # tupla[0] = total de votos no objeto Candidato
            # tupla[1] = total de votos na urna
            if tupla[0].votos != tupla[1]:
                raise ErroUrna('Tentativa de fraudar votação. Candidato {}. Votos válidos={}, Votos do candidato={}.'.
                               format(tupla[0], tupla[1], tupla[0].votos))
            somaVotos += tupla[1]

        # O total de votos deve ser igual ao total de votantes ou outra forma de adulterar a urna foi empregada
        if somaVotos != self._votaram:
            raise ErroUrna('Erro inesperado. Total de votos não confere. Registrados={}, Contabilizados={}'.format(
                self._votaram, somaVotos))

    # Exibição dos resultados
    def exibirResultados(self):
        # Obtem os dados atualizados da urna
        dados = self.resultados
        # Configura o título
        if self.votacaoTerminada:
            titulo = 'Resultado final das eleições- {} votos computados'.format(self._votaram)
        else:
            titulo = 'Resultado PARCIAL das eleições - {} votos computados'.format(self._votaram)
        # Apresenta gráfico de barras
        plt.bar(dados[1], dados[2])
        plt.title(titulo)
        plt.xlabel('Candidatos')
```

```
        plt.ylabel('Votos')
        plt.show()
        # Apresenta dados textuais
        print(dados[0])
        print()

    # Representação visual do objeto
    def __repr__(self):
        # Relação de candidatos
        s = 'URNA\n' + self.strCandidatos
        # Relação de apartamentos votantes
        s += self.strApartamentos
        # Total de votos computados
        s += 'Total de votos computados: {}\n\n'.format(self._votaram)
        # Resultados da votação
        s += self.resultados
        return s


# ---------------------------------------------------------------------------
# Classe Menu
#
# Execução das tarefas de suporte para votação
#
# Atributos:
# - Dicionário de moradores, candidatos e apartamentos
# - Objeto Urna para armazenamento dos votos
#
class Menu:

    # Construtor
    def __init__(self):
        self._moradores = Moradores()   # Dicionário vazio de moradores
        self._urna = None               # Urna não preparada para votação

    # Método utilitário para apresentar títulos em tela para interação com usuário
    def titulo(self, msg):
        qtd_carac = len(msg)
        print()
        print("=" * qtd_carac)
        print(msg)
        print("=" * qtd_carac)

    # Método utilitário para apresentar uma mensagem e aguardar pela tecla Enter
    def _tecleEnter(self, msg=''):
        msg += ' <Tecle Enter para continuar>'
        input(msg)

    # Método utilitário para apresentar uma mensagem e arguardar confirmação do usuário
    def _Confirmar(self, msg=''):
        opcao = ' '
        while opcao not in ['S','N']:
```

```
            opcao = input(msg + ' (S/N)? ').upper()
        return opcao == 'S'

    # Laço do menu principal
    def executar(self):
        opcao = 1
        while opcao > 0:
            self.titulo('MENU PRINCIPAL')
            print('1 - Cadastro de moradores e candidatos')
            print('2 - Configurar urna de votação')
            print('3 - Descartar urna configurada')
            print('4 - Votação')
            print('5 - Resultados')
            print('0 - Encerrar')
            try:
                opcao = int(input('Opção: '))
                if opcao == 1:
                    # Cadastramentos
                    self._menuCadastro()
                elif opcao == 2:
                    # Configurar urna
                    self._configurarUrna()
                elif opcao == 3:
                    # Descartar urna configurada
                    self._descartarUrna()
                elif opcao == 4:
                    # Votação
                    if self._urna is None:
                        self._tecleEnter('Urna ainda não configurada. Votação não iniciada.')
                    else:
                        try:
                            self.__votar()
                        except Exception as e:
                            self._tecleEnter(e.__str__())
                elif opcao == 5:
                    # Resultados até o momento...
                    if self._urna is None:
                        self._tecleEnter('Votação ainda não iniciada.')
                    else:
                        self.titulo('RESULTADOS')
                        try:
                            self._urna.exibirResultados()
                        except Exception as e:
                            print(e.__str__())
                            opcao = 1       # Para continuar no menu
                        self._tecleEnter()
                elif opcao == 0:
                    # Finalizar execução
                    if self._Confirmar('Confirma encerramento'):
                        break
                    else:
                        opcao = 1    # Para continuar no menu
```

```
                else:
                    raise ErroMenu()
            except Exception:
                print('Opção inválida.')

    # Cadastramento de moradores, candidatos e apartamentos
    def _menuCadastro(self):
        opcao = 1
        while opcao > 0:
            self.titulo('MANUTENÇÃO CADASTRAL')
            print('1 - Cadastrar candidato ou morador')
            print('2 - Alterar ou excluir morador ou candidato')
            print('3 - Carregar em lote a partir de arquivo CSV')
            print('4 - Listar candidatos')
            print('5 - Listar apartamentos e moradores')
            print('0 - Voltar')
            try:
                opcao = int(input('Opção: '))

                if opcao > 0 and opcao < 4 and self._urna is not None:
                    # Operações não autorizadas com votação em andamento
                    self._tecleEnter('Não autorizado: Votação iniciada.')
                elif opcao == 1:
                    # Cadastrar morador ou candidato
                    self._cadastrarCandidatoMorador()
                elif opcao == 2:
                    # Alterar ou excluir morador ou candidato
                    self._editarCandidatoMorador()
                elif opcao == 3:
                    # Carregar dados de arquio CSV
                    self._carregarCSV()
                elif opcao == 4:
                    # Listar candidatos
                    self.titulo('LISTA DE CANDIDATOS')
                    self._moradores.exibirCandidatos()
                    self._tecleEnter()
                elif opcao == 5:
                    # Listar apartamentos e moradores
                    self.titulo('LISTA DE APARTAMENTOS E MORADORES')
                    self._moradores.exibir_apartamentos()
                    self._tecleEnter()
                elif opcao == 0:
                    # Voltar ao menu anterior
                    return
                else:
                    # Valor de opção inválida
                    raise ErroMenu()
            except Exception as e:
                print('Opção inválida. '+ e.__str__())

    # Cadastramento de candidato ou morador
    def _cadastrarCandidatoMorador(self):
```

```
        self.titulo("CADASTRAMENTO DE CANDIDATO/MORADOR")
        # Pergunta o tipo de pessoa a cadastrar
        tipo = ' '
        while tipo not in ['C','M','V']:
            tipo = input("Candidato, Morador ou Voltar (C/M/V)? ").upper()

        if tipo == 'V':
            # Retornar ao menu
            return

        # Instancia de acordo com o tipo
        if tipo == 'C':
            morador = Candidato()
        else:
            morador = Morador()

        # Registra morador ou candidato criado
        try:
            self._moradores.registrarMorador(morador)
            msg = 'Cadastramento bem sucedido.'
        except Exception as e:
            msg = e.__str__()
        self._tecleEnter(msg)

    # Alteração ou Exclusão de Candidato ou Morador
    def _editarCandidatoMorador(self):
        apto = 1
        while apto is not None:
            self.titulo('EDIÇÃO DE CANDIDATO/MORADOR')
            # Primeiro passo: Seleção do apartamento
            apto = self._selecaoApto()    # Retorna None se o usuário cancelar a seleção
            morador = 1
            while apto is not None and morador is not None:
                # Segundo passo é a seleção do morador.
                morador = self._selecaoMorador(apto)
                if morador is not None:       # Usuário cancelou a operação - volta para seleção de apartamento
                    # Terceiro passo é a seleção da operação
                    acao = input('Deseja (A)lterar ou (E)xcluir (qualquer outro valor para cancelar)? ').upper()
                    if acao == 'A':
                        # Alterar nome do morador
                        self._alterarCandidatoMorador(morador)
                    elif acao == 'E':
                        # Excluir morador
                        self._excluirCandidatoMorador(apto, morador)
                        if apto.qtdMoradores == 0:
                            self._tecleEnter('Nenhum morador mais no apartamento. Apartamento excluído.')
                            morador = None

    # Alteração do nome do candidato/morador
    def _alterarCandidatoMorador(self, morador):
        self.titulo('ALTERAÇÃO DE CANDIDATO/MORADOR')
        print('Nome atual:', morador.nome)
```

```
        nomeValido = False
        while not nomeValido:
            novoNome = input('Novo nome.: ')
            nomeValido = len(novoNome.strip(' ')) > 0
            if nomeValido and self._Confirmar('Confirma a alteração de nome?'):
                morador.nome = novoNome
                self._tecleEnter('Alteração concluida.')
            else:
                nomeValido = True

    # Exclusão do candidato/morador
    def _excluirCandidatoMorador(self, apartamento, morador):
        self.titulo('EXCLUSÃO DE CANDIDATO/MORADOR')
        if self._Confirmar('Confirma a exclusão do morador "{}"'.format(morador.nome)):
            self._moradores.excluirMorador(morador)
            self._tecleEnter('Exclusão concluida.')

    # Carregar apartamentos, moradores e candidatos a partir de um arquivo CSV
    def _carregarCSV(self):
        self.titulo("CARGA DE ARQUIVO CSV")
        arquivo = input('Caminho e nome do arquivo CSV: ')
        try:
            self._moradores.carregarCSV(arquivo)
            msg = 'Carregamento bem sucedido.'
        except Exception as e:
            msg = e.__str__()
        self._tecleEnter(msg)

    # Configuração da urna para início de votação
    def _configurarUrna(self):
        self.titulo('CONFIGURAÇÃO DA URNA DE VOTAÇÃO')

        # Pelo menos 2 candidatos
        if self._moradores.qtdCandidatos < 2:
            self._tecleEnter('Para iniciar votação são necessários 2 candidatos ou mais.')
            return

        # Pelo menos 2 apartamentos
        if self._moradores.qtdApartamentos < 2:
            self._tecleEnter('Para iniciar votação são necessários 2 apartamentos ou mais.')
            return

        # Registra os candidatos na urna
        if self._urna is None:
            self._urna = Urna()
            for candidato in self._moradores.candidatos.values():
                self._urna.registrarCandidato(candidato)

            # Registra os apartamentos votantes na urna
            for apto in self._moradores.apartamentos.values():
                self._urna.registrarApartamento(apto)
            self._tecleEnter('Configuração bem sucedida.')
```

```
        else:
            self._tecleEnter('Atenção: Urna já estava configurada.')

    # Descarte de urna configurada para votação
    def _descartarUrna(self):
        self.titulo('DESCARTE DE URNA')
        if self._urna is None:
            msg = 'Nenhuma urna ativa. Nada a fazer.'

        elif self._urna.votacaoIniciada:
            msg = 'Votação em andamento. Urna não pode ser descartada.'

        elif self._Confirmar('Confirma descarte da urna'):
            self._urna = None
            msg = 'Urna reinicializada.'

        else:
            msg = 'Descarte cancelado. Urna continua ativa para início de votação.'
        self._tecleEnter(msg)

    # Votação
    def __votar(self):
        # Executa loop até todos que todos os apartamentos tenham votado
        while not self._urna.votacaoTerminada:
            self.titulo('VOTAÇÃO')
            # Antes do primeiro voto alerta que a urna não pode mais ser alterada
            if not (self._urna.votacaoIniciada or
                    self._Confirmar('Se a votação for iniciada não poderá mais ser interrompida. Confirma')):
                return

            # Primeiro passo da votação: Seleção do apartamento do votante
            apto = self._selecaoApto()
            if apto is None:
                # Sair do loop de votação e voltar ao menu principal
                # Votação pode ser retomada normalmente depois
                break

            # Não permite que um apartamento vote mais que uma vez
            if self._urna.apartamentoVotou(apto.numero):
                self._tecleEnter('Esse apartamento já votou.')
                continue

            # ALTERNATIVA: Utilizar o primeiro objeto Morador do objeto Apartamento, pois como testamos previamente
            # contra duplicidade de votação e saber quem está votando do apartamento não é relevante.
            apto.morador(0).votar(self._urna)

            # ALTERNATIVA: Exibir os moradores e permitir que o votante se identifique
            #
            # Segundo passo é a seleção do morador.
            # Se retornar None o usuário selecionou "cancelar" e o program deve voltar para a seleção de apartamento
            # do votante
            #morador = self._selecaoMorador(apto)
```

```
            #if morador is not None:
            #    # Terceiro passo é a seleção do candidato (Voto)
            #    morador.votar(self._urna)

            # Aguarda alguns segundos antes de preparar para o próximo voto, para o morador visualizar a confirmação
            # de voto registrado
            if self._urna.votacaoTerminada:
                print('* * * Votação Encerrada * * *')
            else:
                print('Preparando para próximo voto...')
            time.sleep(2)

    # Seleção do apartamento onde mora a pessoa que irá votar
    def _selecaoApto(self):
        apto = None
        while apto is None:
            numApto = input('Informe o número do apartamento (0 para cancelar): ')
            try:
                numApto = int(numApto)
                # Usuário deseja cancelar e voltar
                if numApto == 0:
                    break
                # Seleciona o apartamento
                apto = self._moradores.apartamento(numApto)
            except:
                print('Apartamento inválido!')
        return apto

    # Seleção do obter_morador que irá votar
    def _selecaoMorador(self, apto):
        morador = None
        while morador is None:
            apto.exibirMoradores()
            try:
                num = int(input('Selecione morador (0 para cancelar): '))
                if num < 0 or num > apto.qtdMoradores:
                    raise Exception()
                # Usuário deseja cancelar e voltar
                if num == 0:
                    return None
                # Seleciona o objeto Morador correspondente ao número escolhido
                morador = apto.morador(num-1)
            except:
                self._tecleEnter('Número de morador inválido.')
        return morador


# -----------------------------------------------------------------------------
# Programa Principal
#
menu = Menu()
menu.executar()
```

```
==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

================
DESCARTE DE URNA
================

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==============================
CONFIGURAÇÃO DA URNA DE VOTAÇÃO
==============================
```

```
==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

====================
CARGA DE ARQUIVO CSV
====================

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

====================
CARGA DE ARQUIVO CSV
====================

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar
```

```
====================
LISTA DE CANDIDATOS
====================
Candidatos - quantidade = 3
Candidato 0 - Morador Joao, Apto.1 - Votos: 0
Candidato 0 - Morador Daniela, Apto.21 - Votos: 0
Candidato 0 - Morador Alexandre, Apto.43 - Votos: 0


====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar
```

```
================================
LISTA DE APARTAMENTOS E MORADORES
================================
Apartamentos - quantidade = 24
Apto=1
Moradores:
   1) Joao
   2) Maria
Apto=2
Moradores:
   1) Jose
Apto=3
Moradores:
   1) Angelo
   2) Clarisse
Apto=4
Moradores:
   1) Pedro
   2) Marcia
Apto=11
Moradores:
   1) Sergio
   2) Fabiana
Apto=12
Moradores:
   1) Mauro
   2) Neide
Apto=13
Moradores:
   1) Clara
Apto=14
Moradores:
   1) Benedito
   2) Benedita
Apto=21
Moradores:
   1) Carlos
   2) Daniela
Apto=22
Moradores:
   1) Fabiano
Apto=23
Moradores:
   1) Gessica
   2) Francine
Apto=24
Moradores:
   1) Paulo
   2) Roberta
Apto=31
Moradores:
   1) Marcelo
```

```
   2) Paula
Apto=32
Moradores:
   1) Paulo
   2) Paula
Apto=33
Moradores:
   1) Leo
   2) Vania
Apto=34
Moradores:
   1) Elcio
   2) Hilda
Apto=41
Moradores:
   1) Marcos
   2) Denise
Apto=42
Moradores:
   1) Joao
   2) Quiteria
Apto=43
Moradores:
   1) Alexandre
   2) Marilia
Apto=44
Moradores:
   1) Barbosa
   2) Neide
   3) Jessica
   4) Willian
Apto=51
Moradores:
   1) Washington
Apto=52
Moradores:
   1) Felipe
   2) Luciana
Apto=53
Moradores:
   1) Cesar
   2) Fatima
Apto=54
Moradores:
   1) Wilson
   2) Marcia
```

```
====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

============================
EDIÇÃO DE CANDIDATO/MORADOR
============================

Moradores:
   1) Alexandre
   2) Marilia


==============================
ALTERAÇÃO DE CANDIDATO/MORADOR
==============================
Nome atual: Alexandre

Moradores:
   1) Anderson
   2) Marilia


============================
EDIÇÃO DE CANDIDATO/MORADOR
============================

Moradores:
   1) Joao
   2) Maria


==============================
EXCLUSÃO DE CANDIDATO/MORADOR
==============================

Moradores:
   1) Maria


==============================
EXCLUSÃO DE CANDIDATO/MORADOR
==============================

============================
EDIÇÃO DE CANDIDATO/MORADOR
============================

Apartamento inválido!
```

```
====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==================================
CADASTRAMENTO DE CANDIDATO/MORADOR
==================================

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==================================
CADASTRAMENTO DE CANDIDATO/MORADOR
==================================

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar
```

```
================================
LISTA DE APARTAMENTOS E MORADORES
================================
Apartamentos - quantidade = 24
Apto=1
Moradores:
   1) Joao
   2) Maria
Apto=2
Moradores:
   1) Jose
Apto=3
Moradores:
   1) Angelo
   2) Clarisse
Apto=4
Moradores:
   1) Pedro
   2) Marcia
Apto=11
Moradores:
   1) Sergio
   2) Fabiana
Apto=12
Moradores:
   1) Mauro
   2) Neide
Apto=13
Moradores:
   1) Clara
Apto=14
Moradores:
   1) Benedito
   2) Benedita
Apto=21
Moradores:
   1) Carlos
   2) Daniela
Apto=22
Moradores:
   1) Fabiano
Apto=23
Moradores:
   1) Gessica
   2) Francine
Apto=24
Moradores:
   1) Paulo
   2) Roberta
Apto=31
Moradores:
   1) Marcelo
```

```
   2) Paula
Apto=32
Moradores:
   1) Paulo
   2) Paula
Apto=33
Moradores:
   1) Leo
   2) Vania
Apto=34
Moradores:
   1) Elcio
   2) Hilda
Apto=41
Moradores:
   1) Marcos
   2) Denise
Apto=42
Moradores:
   1) Joao
   2) Quiteria
Apto=43
Moradores:
   1) Anderson
   2) Marilia
Apto=44
Moradores:
   1) Barbosa
   2) Neide
   3) Jessica
   4) Willian
Apto=51
Moradores:
   1) Washington
Apto=52
Moradores:
   1) Felipe
   2) Luciana
Apto=53
Moradores:
   1) Cesar
   2) Fatima
Apto=54
Moradores:
   1) Wilson
   2) Marcia
```

```
====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

================================
CONFIGURAÇÃO DA URNA DE VOTAÇÃO
================================

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar
```

```
====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==================
LISTA DE CANDIDATOS
==================
Candidatos - quantidade = 2
Candidato 1 - Morador Daniela, Apto.21 - Votos: 0
Candidato 2 - Morador Anderson, Apto.43 - Votos: 0


====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar
```

```
================================
LISTA DE APARTAMENTOS E MORADORES
================================
Apartamentos - quantidade = 24
Apto=1
Moradores:
   1) Joao
   2) Maria
Apto=2
Moradores:
   1) Jose
Apto=3
Moradores:
   1) Angelo
   2) Clarisse
Apto=4
Moradores:
   1) Pedro
   2) Marcia
Apto=11
Moradores:
   1) Sergio
   2) Fabiana
Apto=12
Moradores:
   1) Mauro
   2) Neide
Apto=13
Moradores:
   1) Clara
Apto=14
Moradores:
   1) Benedito
   2) Benedita
Apto=21
Moradores:
   1) Carlos
   2) Daniela
Apto=22
Moradores:
   1) Fabiano
Apto=23
Moradores:
   1) Gessica
   2) Francine
Apto=24
Moradores:
   1) Paulo
   2) Roberta
Apto=31
Moradores:
   1) Marcelo
```

```
   2) Paula
Apto=32
Moradores:
   1) Paulo
   2) Paula
Apto=33
Moradores:
   1) Leo
   2) Vania
Apto=34
Moradores:
   1) Elcio
   2) Hilda
Apto=41
Moradores:
   1) Marcos
   2) Denise
Apto=42
Moradores:
   1) Joao
   2) Quiteria
Apto=43
Moradores:
   1) Anderson
   2) Marilia
Apto=44
Moradores:
   1) Barbosa
   2) Neide
   3) Jessica
   4) Willian
Apto=51
Moradores:
   1) Washington
Apto=52
Moradores:
   1) Felipe
   2) Luciana
Apto=53
Moradores:
   1) Cesar
   2) Fatima
Apto=54
Moradores:
   1) Wilson
   2) Marcia
```

```
====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

================
DESCARTE DE URNA
================

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

===========================
EDIÇÃO DE CANDIDATO/MORADOR
===========================

Moradores:
   1) Joao
   2) Maria
```

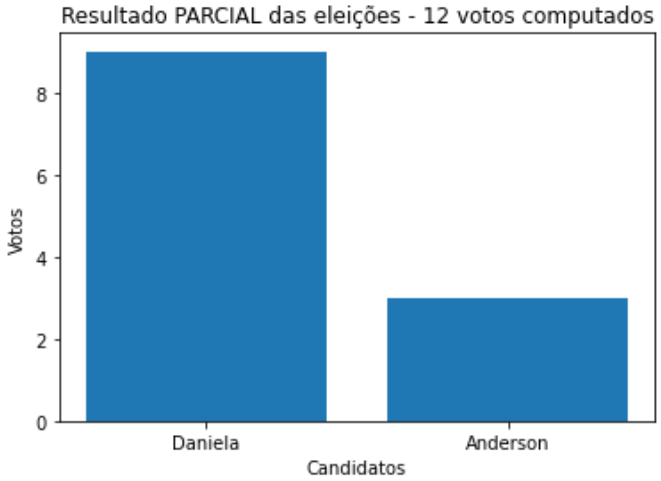
```
==============================
ALTERAÇÃO DE CANDIDATO/MORADOR
==============================
Nome atual: Joao

Moradores:
   1) Jose
   2) Maria


============================
EDIÇÃO DE CANDIDATO/MORADOR
============================

====================
MANUTENÇÃO CADASTRAL
====================
1 - Cadastrar candidato ou morador
2 - Alterar ou excluir morador ou candidato
3 - Carregar em lote a partir de arquivo CSV
4 - Listar candidatos
5 - Listar apartamentos e moradores
0 - Voltar

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==============================
CONFIGURAÇÃO DA URNA DE VOTAÇÃO
==============================

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

=======
VOTAÇÃO
=======
```

```
==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======
```

```
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

=======
VOTAÇÃO
=======

Apartamento inválido!

Apartamento inválido!

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)
```

```
* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


Candidato inválido
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


Candidato inválido
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)
```

```
* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==========
RESULTADOS
==========
```

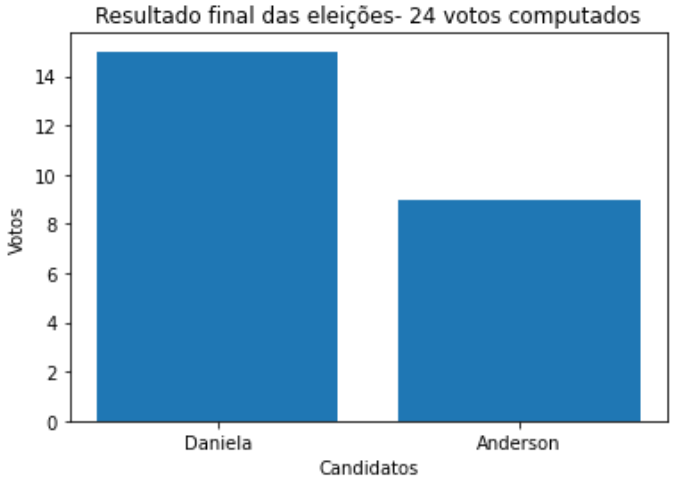



```
Votação em andamento. Quantidades PARCIAIS

RESULTADOS:
1 - Daniela = 9 votos
2 - Anderson = 3 votos



==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======
```

```
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======
```

```
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======
```

```
Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

Preparando para próximo voto...

=======
VOTAÇÃO
=======

Candidatos: 2
1 - Daniela (Apto 21)
2 - Anderson (Apto 43)


* * * Voto registrado com sucesso * * *.

* * * Votação Encerrada * * *

==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar

==========
RESULTADOS
==========
```




```
Votação terminada.

RESULTADOS:
1 - Daniela = 15 votos
2 - Anderson = 9 votos



==============
MENU PRINCIPAL
==============
1 - Cadastro de moradores e candidatos
2 - Configurar urna de votação
3 - Descartar urna configurada
4 - Votação
5 - Resultados
0 - Encerrar
```

In [ ]: