src
Linguagem de Programação I - Entrega 1 - Artur de Oliveira Fonseca

controle - desempenho_partidas_xadrez

```
from util.gerais import mostrar_objetos
from entidades.jogador import *
from entidades.ritmojogo import *
from entidades.organizacao_rating import *

def cadastro_jogadores():
    inserir_jogador(Jogador(nome='Rafael Correa Viana',
rating_atual=2435, estrangeiro=False))
    inserir_jogador(Jogador('Magnus Carlsen', 2862, True))
    inserir_jogador(Jogador('Paulo Fonseca', 1850, False))
    inserir_jogador(Jogador('Artur Fonseca', 2700, False))
    inserir_jogador(Jogador('Sara Sakai', 2000, True))
    inserir_jogador(Jogador('Tigran Petrosian', 2652, True))
    inserir_jogador(Jogador('Mikhail Tal', 2700, True))
    inserir_jogador(Jogador('Hikaru Nakamura', 2795, True))


def cadastro_organizacoes():
    inserir_organizacoes(OrganizacaoRating(
        nome='FIDE', unidade_federativa='MS', cidade='Dourados'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='SP', cidade='Presidente
Prudente'))
    inserir_organizacoes(OrganizacaoRating(
        nome='CBX', unidade_federativa='SP', cidade='São Paulo'))
    inserir_organizacoes(OrganizacaoRating(
        nome='FIDE', unidade_federativa='MG', cidade='Belo
Horizonte'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='SP', cidade='Barueri'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='MS', cidade='Campo
Grande'))


def cadastro_ritmo_jogo():
    inserir_ritmos_de_jogo(RitmoJogo(tempo_inicial='5:00',
acrescimo=5))
    inserir_ritmos_de_jogo(RitmoJogo('10:00', 0))
    inserir_ritmos_de_jogo(RitmoJogo('1:00', 1))
    inserir_ritmos_de_jogo(RitmoJogo('12:00', 2))
    inserir_ritmos_de_jogo(RitmoJogo('21:00', 0))
    inserir_ritmos_de_jogo(RitmoJogo('30:00', 15))
    inserir_ritmos_de_jogo(RitmoJogo('0:30', 2))
```

```
if __name__ == '__main__':
    # cadastro dos jogadores
    cadastro_jogadores()
    cabecalho = 'Lista de Jogadores: Nome | Rating | Título |
Estrangeiro'
    lista, filtro = selecionar_jogadores()
    mostrar_objetos(cabecalho=cabecalho, lista=lista,
filtros=filtro)
    print('\n')
    lista, filtro = selecionar_jogadores(rating_minimo=1900)
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_jogadores(rating_minimo=1900,
titulo_fide_min='GM')
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_jogadores(rating_minimo=1900,
titulo_fide_min='GM', estrangeiro = True)
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_jogadores(rating_minimo=1900,
titulo_fide_min='GM', estrangeiro = True, prefixo_nome ='H')
    mostrar_objetos(cabecalho, lista, filtro)

    # cadastro das organizações
    cadastro_organizacoes()
    cabecalho = 'Lista de Organizações: Nome -- Unidade Federativa
-- Cidade'
    lista, filtro = selecionar_organizacoes()
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_organizacoes(nome='LBX')
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_organizacoes('LBX',
unidade_federativa='SP')
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_organizacoes('LBX', 'SP',
cidade='Barueri')
    mostrar_objetos(cabecalho, lista, filtro)

    # cadastro do ritmo de jogo
    cadastro_ritmo_jogo()
    cabecalho = 'Lista de Ritmos de Jogo: Tempo Inicial --
Acréscimo'
```

```
    lista, filtro = selecionar_ritmos_de_jogo()
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro =
selecionar_ritmos_de_jogo(tempo_minimo_inicial='1:00')
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_ritmos_de_jogo('1:00',
tempo_maximo_inicial='10:00')
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_ritmos_de_jogo('1:00', '10:00',
acrescimo_maximo=0)
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
```

entidade - ritmojogo

```
from util.gerais import mostrar_objetos

class RitmoJogo:
    def __init__(self, tempo_inicial, acrescimo):
        """
        Inicializa um objeto RitmoJogo com o tempo inicial e o
acréscimo especificados.

        :param tempo_inicial: O tempo inicial do jogo no formato
'minutos:segundos' ou 'minutos'.
        :param acrescimo: O acréscimo de tempo para cada jogada.
        """
        self.tempo_inicial = self.modificar_tempo(tempo_inicial)
        self.acrescimo = acrescimo

    @staticmethod
    def modificar_tempo(tempo):
        """
        Método estático que modifica o tempo para o formato de
segundos.

        :param tempo: O tempo no formato 'minutos:segundos' ou
'minutos'.
        :return: O tempo convertido para segundos.
        """
        try:
            if ':' in tempo:
                minutos, segundos = map(int, tempo.split(':'))
                return minutos * 60 + segundos
```

```
            return int(tempo)
        except ValueError:
            print("Formato de tempo inválido. Use o formato
'minutos:segundos' ou 'minutos'.")
            return None

    def formatar_tempo_inicial(self):
        """
        Formata o tempo inicial para o formato 'minutos:segundos'.

        :return: O tempo inicial formatado.
        """
        minutos = self.tempo_inicial // 60
        segundos = self.tempo_inicial % 60
        return f'{minutos:02d}:{segundos:02d}'

    def __str__(self):
        """
        Retorna uma representação em string do objeto RitmoJogo.

        :return: A representação em string do objeto.
        """
        tempo_formatado = self.formatar_tempo_inicial()
        formato = '|{:^7}|{:^5}|'
        return formato.format(tempo_formatado, '+' +
str(self.acrescimo))


ritmos_de_jogo = []


def obter_ritmos_de_jogo():
    """
    Retorna uma lista contendo todos os ritmos de jogo inseridos.

    :return: A lista de ritmos de jogo.
    """
    return ritmos_de_jogo


def inserir_ritmos_de_jogo(ritmo_de_jogo):
    """
    Insere um ritmo de jogo na lista de ritmos.

    :param ritmo_de_jogo: O ritmo de jogo a ser inserido.
    """
    ritmos_de_jogo.append(ritmo_de_jogo)
```

```
def _filtrar_por_tempo_inicial_minimo(ritmo_de_jogo,
tempo_minimo_inicial):
    """
    Função interna para filtrar os ritmos de jogo com base no tempo
inicial mínimo.

    :param ritmo_de_jogo: O ritmo de jogo a ser filtrado.
    :param tempo_minimo_inicial: O tempo mínimo inicial desejado.
    :return: True se o ritmo de jogo atender ao critério de filtro,
False caso contrário.
    """
    if tempo_minimo_inicial is None:
        return True
    tempo_minimo_segundos =
RitmoJogo.modificar_tempo(tempo_minimo_inicial)
    return ritmo_de_jogo.tempo_inicial >= tempo_minimo_segundos


def _filtrar_por_acrescimo_maximo(ritmo_de_jogo,
acrescimo_maximo):
    """
    Função interna para filtrar os ritmos de jogo com base no
acréscimo máximo.

    :param ritmo_de_jogo: O ritmo de jogo a ser filtrado.
    :param acrescimo_maximo: O acréscimo máximo desejado.
    :return: True se o ritmo de jogo atender ao critério de filtro,
False caso contrário.
    """
    return acrescimo_maximo is None or ritmo_de_jogo.acrescimo <=
acrescimo_maximo


def _filtrar_por_tempo_inicial_maximo(ritmo_de_jogo,
tempo_maximo_inicial):
    '''
    Função interna para filtrar os ritmos de jogo com base no tempo
máximo inicial.
    :param ritmo_de_jogo:
    :param tempo_maximo_inicial:
    :return:
    '''
    if tempo_maximo_inicial is None:
        return True
    tempo_maximo_segundos =
RitmoJogo.modificar_tempo(tempo_maximo_inicial)
    return ritmo_de_jogo.tempo_inicial <= tempo_maximo_segundos
```

```
def _adicionar_filtro(tempo_minimo_inicial=None,
tempo_maximo_inicial=None, acrescimo_maximo=None):
    """
    Adiciona filtros aplicados à seleção de ritmos de jogo.

    :param tempo_minimo_inicial: O tempo mínimo inicial desejado.
    :param acrescimo_maximo: O acréscimo máximo desejado.
    :return: Uma string representando os filtros aplicados.
    """
    filtros_dict = {
        'Tempo Mínimo Inicial': tempo_minimo_inicial,
        'Tempo Máximo Inicial': tempo_maximo_inicial,
        'Acréscimo Máximo': acrescimo_maximo
    }
    str_filtros = 'Filtros -- '
    for key, filtro in filtros_dict.items():
        if filtro is not None:
            str_filtros += f'{key}: {filtro} -- '
    return str_filtros.rstrip(' -- ')


def selecionar_ritmos_de_jogo(tempo_minimo_inicial=None,
tempo_maximo_inicial=None, acrescimo_maximo=None):
    """
    Seleciona os ritmos de jogo com base nos filtros especificados.

    :param tempo_minimo_inicial: O tempo mínimo inicial desejado.
    :param tempo_maximo_inicial: O tempo máximo inicial desejado.
    :param acrescimo_maximo: O acréscimo máximo desejado.
    :return: Uma lista dos ritmos de jogo selecionados e uma string
representando os filtros aplicados.
    """
    ritmos_selecionados = []
    str_filtros = _adicionar_filtro(tempo_minimo_inicial,
tempo_maximo_inicial, acrescimo_maximo)
    filtros = [
        lambda ritmo: _filtrar_por_tempo_inicial_minimo(
            ritmo_de_jogo=ritmo,
            tempo_minimo_inicial=tempo_minimo_inicial
        ),
        lambda ritmo: _filtrar_por_acrescimo_maximo(
            ritmo_de_jogo=ritmo,
            acrescimo_maximo=acrescimo_maximo
        ),
        lambda ritmo: _filtrar_por_tempo_inicial_maximo(
            ritmo_de_jogo=ritmo,
```

```
            tempo_maximo_inicial=tempo_maximo_inicial
        )
    ]
    for ritmo in ritmos_de_jogo:
        if all(filtro(ritmo) for filtro in filtros):
            ritmos_selecionados.append(ritmo)
    return ritmos_selecionados, str_filtros



if __name__ == '__main__':
    cabecalho = 'Lista de Ritmos de Jogo: Tempo Inicial --
Acréscimo'
    inserir_ritmos_de_jogo(RitmoJogo(tempo_inicial='5:00',
acrescimo=0))
    inserir_ritmos_de_jogo(RitmoJogo('10:00', 0))
    inserir_ritmos_de_jogo(RitmoJogo('1:00', 1))
    inserir_ritmos_de_jogo(RitmoJogo('12:00', 2))
    inserir_ritmos_de_jogo(RitmoJogo('21:00', 0))
    inserir_ritmos_de_jogo(RitmoJogo('30:00', 15))
    lista, filtro = selecionar_ritmos_de_jogo()
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista,filtro =
selecionar_ritmos_de_jogo(tempo_minimo_inicial='5:00',
tempo_maximo_inicial='10:00')
    mostrar_objetos(cabecalho, lista, filtro)
```

entidade - jogador

```
from util.gerais import mostrar_objetos


class Jogador:
    def __init__(self, nome, rating_atual, estrangeiro=False):
        self.nome = nome
        self.rating_atual = rating_atual
        # Setter faz a atribuição correta de self.titulo_fide
        # com base na variável rating_atual
        self.titulo_fide = ''
        # fim do comentário
        self.estrangeiro = estrangeiro

    def __str__(self):
        formato = '{:<20} | {:<4} | {:^3} | {:^14} |'
        return formato.format(self.nome, self.rating_atual,
self.titulo_fide
        , 'Estrangeiro' if self.estrangeiro else '---')
```

```python
    @property
    def titulo_fide(self):
        return self._titulo_fide

    # Setter responsável pela atribuição e validação do
self.titulo_fide
    @titulo_fide.setter
    def titulo_fide(self, new_value):
        if self.rating_atual >= 2500:
            self._titulo_fide = 'GM'
            return
        if self.rating_atual >= 2400:
            self._titulo_fide = 'IM'
            return
        if self.rating_atual >= 2300:
            self._titulo_fide = 'FM'
            return
        if self.rating_atual >= 2100:
            self._titulo_fide = 'CM'
            return
        self._titulo_fide = '--'


# fim da classe


jogadores = []


def obter_jogadores():
    """
    Retorna uma lista contendo todos os jogadores inseridos
    pela função inserir_jogadores().
    :return:
    """
    return jogadores


def inserir_jogador(jogador):
    """
    Insere um jogador numa lista.
    A lista deve ser recuperada a partir da função obter_jogadores()
    :param jogador:
    """
    jogadores.append(jogador)
```

```python
def _filtrar_por_rating(jogador, rating_minimo):
    """
        Função para uso interno ajudando o funcionamento da função
seleciona jogadores.
        Não é necessário utilizar.
        :param jogador:
        :param rating_minimo:
        :return:
        """
    return rating_minimo is None or jogador.rating_atual >=
rating_minimo


def _filtrar_por_prefixo_nome(jogador, prefixo_nome):
    """
        Função para uso interno ajudando o funcionamento da função
seleciona jogadores.
        Não é necessário utilizar.
        :param jogador:
        :param prefixo_nome:
        :return:
        """
    return prefixo_nome is None or
jogador.nome.startswith(prefixo_nome)


def _filtrar_por_titulo_fide(jogador, titulo_fide_minimo):
    """
        Função para uso interno ajudando o funcionamento da função
seleciona jogadores.
        Não é necessário utilizar.
        :param jogador:
        :param titulo_fide_minimo:
        :return:
        """
    if titulo_fide_minimo is None:
        return True
    titulo_numerico = {'Sem Título': 0, 'CM': 1, 'FM': 2, 'IM': 3,
'GM': 4}
    jogador_titulo_numerico =
titulo_numerico.get(jogador.titulo_fide, 0)
    return jogador_titulo_numerico >=
titulo_numerico.get(titulo_fide_minimo, 0)


def _filtrar_por_estrangeiro(jogador, estrangeiro):
    """
```

```
    Função para uso interno ajudando o funcionamento da função
seleciona jogadores.
    Não é necessário utilizar.
    :param jogador:
    :param estrangeiro:
    :return:
    """
    return estrangeiro is None or jogador.estrangeiro ==
estrangeiro


def _adicionar_filtro(prefixo_nome=None, rating_minimo=None,
titulo_fide_min=None, estrangeiro=None):
    """
    Função para uso interno ajudando o funcionamento da função
seleciona jogadores.
    Não é necessário utilizar.
    :param prefixo_nome:
    :param rating_minimo:
    :param titulo_fide_min:
    :param estrangeiro:
    :return:
    """
    filtros_dict = {
        'Prefixo Nome': prefixo_nome,
        'Mínimo de Rating': rating_minimo,
        'Título Fide Mínimo': titulo_fide_min,
        'Estrangeiro': estrangeiro
    }
    str_filtros = 'Filtros -- '
    for filtro, valor in filtros_dict.items():
        if valor is not None:
            str_filtros += f' {filtro}: {valor} -- '
    return str_filtros.rstrip(' --')


def selecionar_jogadores(prefixo_nome: str = None, rating_minimo:
int = None,
                         titulo_fide_min: str = None, estrangeiro:
bool = None) -> list and str:
    """
    Retorna uma lista de jogadores com base nos parâmetros definidos.
    Deve ser utilizada em conjunto com a função inserir_jogador() e
obter_jogador()
    :param prefixo_nome:
    :param rating_minimo:
    :param titulo_fide_min:
    :param estrangeiro:
```

```
    :return:
    """
    jogadores_selecionados = []
    str_filtros = _adicionar_filtro(prefixo_nome, rating_minimo,
titulo_fide_min, estrangeiro)
    filtros = [
        lambda jogador: _filtrar_por_rating(jogador,
rating_minimo),
        lambda jogador: _filtrar_por_prefixo_nome(jogador,
prefixo_nome),
        lambda jogador: _filtrar_por_titulo_fide(jogador,
titulo_fide_min),
        lambda jogador: _filtrar_por_estrangeiro(jogador,
estrangeiro)
    ]
    for jogador in jogadores:
        if all(filtro(jogador) for filtro in filtros):
            jogadores_selecionados.append(jogador)
    return jogadores_selecionados, str_filtros


if __name__ == '__main__':
    cabecalho = 'Lista de Jogadores: Nome | Rating | Título |
Estrangeiro'
    inserir_jogador(Jogador(nome='Rafael Correa Viana',
rating_atual=2435, estrangeiro=False ))
    inserir_jogador(Jogador('Magnus Carlsen', 2862,  True))
    inserir_jogador(Jogador('Paulo Fonseca', 1850,  False))
    inserir_jogador(Jogador('Artur Fonseca', 2700, False))
    inserir_jogador(Jogador('Sara Sakai', 2000, True))
    inserir_jogador(Jogador('Tigran Petrosian', 2652, True))
    inserir_jogador(Jogador('Mikhail Tal', 2700, True))
    inserir_jogador(Jogador('Hikaru Nakamura', 2795, True))
    lista, filtro = selecionar_jogadores()
    mostrar_objetos(cabecalho=cabecalho, lista=lista,
filtros=filtro)
    print('\n')
    lista, filtro = selecionar_jogadores(rating_minimo=1900,
titulo_fide_min='GM', estrangeiro=True, prefixo_nome='H')
    mostrar_objetos(cabecalho, lista, filtro)
```

entidade - organizacao_rating

```
from util.gerais import mostrar_objetos


class OrganizacaoRating:

    def __init__(self, nome, cidade, unidade_federativa):
```

```
        self.nome = nome
        self.cidade = cidade
        self.unidade_federativa = unidade_federativa

    def __str__(self):
        formato = '{:^6} | {:^4} | {:<20}'
        return formato.format(self.nome, self.unidade_federativa,
self.cidade)

    # getter para a variável nome
    @property
    def nome(self):
        return self._nome

    # setter responsável por garantir que o nome da organização
esteja dentro das existentes
    @nome.setter
    def nome(self, new_value):
        new_value_upper = new_value.upper()
        if new_value_upper in ['FIDE', 'CBX', 'LBX']:
            self._nome = new_value_upper
            return
        self._nome = 'FIDE'

    @property
    def unidade_federativa(self):
        return self._unidade_federativa

    # setter responsável por garantir que a self.unidade_federativa
esteja
    # entre as ufs existentes, e se existir, converte para
maiúscula
    @unidade_federativa.setter
    def unidade_federativa(self, new_value):
        new_value_upper = new_value.upper()
        estados = [
            'AC', 'AL', 'AP', 'AM', 'BA',
            'CE', 'DF', 'ES', 'GO', 'MA',
            'MT', 'MS', 'MG', 'PA', 'PB',
            'PR', 'PE', 'PI', 'RJ', 'RN',
            'RS', 'RO', 'RP', 'SC', 'SP',
            'SE', 'TO'
        ]
        if new_value_upper in estados:
            self._unidade_federativa = new_value_upper
            return
        self._unidade_federativa = '--'
```

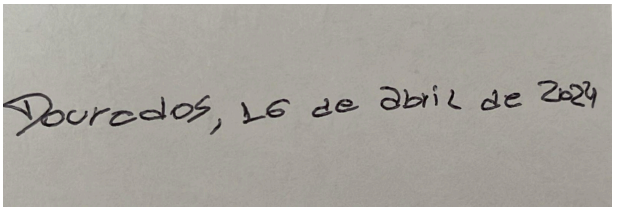
```python
organizacoes = []


def obter_orgarnizacoes():
    return organizacoes


def inserir_organizacoes(organizacao):
    organizacoes.append(organizacao)


def _filtrar_por_estado(organizacao, uf_filtro):
    """
    Função para uso interno ajudando o funcionamento da função
    seleciona_organizações. Não é necessário utilizar
    :param organizacao:
    :param uf_filtro:
    :return:
    """
    return uf_filtro is None or organizacao.unidade_federativa ==
uf_filtro


def _filtrar_por_cidade(organizacao, cidade_filtro):
    '''
    Função para uso interno ajudando o funcionamento da função
    seleciona_organizações. Não é necessário utilizar
    :param organizacao:
    :param cidade_filtro:
    :return:
    '''
    return cidade_filtro is None or organizacao.cidade ==
cidade_filtro


def _filtrar_por_nome(organizacao, nome_filtro):
    '''
    Função para uso interno ajudando o funcionamento da função
    seleciona_organizações. Não é necessário utilizar
    :param organizacao:
    :param nome_filtro:
    :return:
    '''
    return nome_filtro is None or organizacao.nome == nome_filtro
```

```
def _adicionar_filtros(nome=None, unidade_federativa=None,
cidade=None):
    '''
    Função para uso interno ajudando o funcionamento da função
    seleciona_organizações. Não é necessário utilizar
    :param nome:
    :param unidade_federativa:
    :param cidade:
    :return:
    '''
    filtros_dict = {
        'Nome': nome,
        'Unidade Federativa': unidade_federativa,
        'Cidade': cidade
    }
    str_filtros = 'Filtros -- '
    for key, filtro in filtros_dict.items():
        if filtro is not None:
            str_filtros += f'{key}: {filtro} -- '
    return str_filtros.rstrip('-- ')


def selecionar_organizacoes(nome=None, unidade_federativa=None,
cidade=None):
    organizacoes_selecionadas = []
    str_filtros = _adicionar_filtros(nome, unidade_federativa,
cidade)
    filtros = [
        lambda organizacao: _filtrar_por_estado(
            organizacao=organizacao,
            uf_filtro=unidade_federativa),
        lambda organizacao: _filtrar_por_cidade(
            organizacao=organizacao,
            cidade_filtro=cidade),
        lambda organizacao: _filtrar_por_nome(
            organizacao=organizacao,
            nome_filtro=nome)
    ]
    for organizacao in organizacoes:
        if all(filtro(organizacao) for filtro in filtros):
            organizacoes_selecionadas.append(organizacao)
    return organizacoes_selecionadas, str_filtros


if __name__ == '__main__':
    cabecalho = 'Lista de Organizações: Nome -- Unidade Federativa
-- Cidade'
    inserir_organizacoes(OrganizacaoRating(
```

```
        nome='FIDE', unidade_federativa='MS', cidade='Dourados'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='SP', cidade='Presidente
Prudente'))
    inserir_organizacoes(OrganizacaoRating(
        nome='CBX', unidade_federativa='SP', cidade='São Paulo'))
    inserir_organizacoes(OrganizacaoRating(
        nome='FIDE', unidade_federativa='MG', cidade='Belo
Horizonte'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='SP', cidade='Barueri'))
    inserir_organizacoes(OrganizacaoRating(
        nome='LBX', unidade_federativa='MS', cidade='Campo
Grande'))
    lista, filtro = selecionar_organizacoes()
    mostrar_objetos(cabecalho, lista, filtro)
    print('\n')
    lista, filtro = selecionar_organizacoes(nome='LBX',
unidade_federativa='SP', cidade='Barueri')
    mostrar_objetos(cabecalho, lista, filtro)
```

util - gerais

```
def mostrar_objeto(indice, objeto) -> None:
    separador = '-'
    ordem = indice + 1
    frase = f'{ordem} {separador} {objeto}'
    print(frase)


def mostrar_objetos(cabecalho, lista, filtros=None) -> None:
    if filtros is not None:
        print(filtros)
    print(cabecalho)
    for indice, objeto in enumerate(lista):
        mostrar_objeto(indice, str(objeto))
```



Carlos de Oliveira Gomes