Atributos

```
"""
POO - Atributos

Atributos -> Representa as caracteristicas do objeto. ou seja, pelos
atributos
nós conseguimos representar computacionalmente os estados de um
objeto.

Em Python, dividimos os atributos em 3 grupos:
    - Atributos de Instância;
    - Atributos de Classe;
    - Atributos Dinâmico;

# Atributos de instância: são atributos declarados dentro do método
construtor.

# OBS: método construtor: È um método especial utilizado para a
construção do objeto.

# Em Java, uma classe Lâmpada, incluindo seus atributos ficaria mais
ou menos:

public class Lampada(){
    private int voltagem;
    private String cor;
    private Boolean ligado = false;

    public Lampada(int voltagem, String cor){
        this.voltagem = voltagem;
        this.cor = cor;
        }
}

class Lampada:
    def __init__(self, voltagem, cor):
        self.voltagem = voltagem
        self.cor = cor
        self.ligado = False

class ContaCorrente:
    def __init__(self, numero, limite, saldo):
        self.numero = numero
        self.limite = limite
        self.saldo = saldo

class Produto:

    def __init__(self, nome, descricao, valor):
        self.nome = nome
        self.descricao = descricao
        self.valor = valor


a = Produto('carlos', 2332, 233)
print(f'nome:{a.nome} descrição:{a.descricao} valor:{a.valor}')

se eu acesso uma variavel com dois anderlaine e pq esse atributo e
privado
exemplo:
```

```
self.__numerador = num

"""

# criando uma classe fração

class Fracao:

    def __init__(self, num, den):
        self.numerador = num
        if den == 0:
            self.denomidor = 1
        else:
            self.denominador = den

    def inverter(self):
       return Fracao(self.denominador, self.numerador)

    def negar(self):
        return Fracao(-self.numerador, self.denominador)

    def simplificador(self):
        pass

a = Fracao(4, 5)
b = Fracao(-2, 7)

c = b.inverter()

print(f'{c.numerador} {c.denominador}')
```

Classes python

```
""" criando uma classe em python"""
class Carro:
    """construtor da classe criada:
    obs: para criarmos o construtor da classe precisamos utilizar a
palavra reservada __init__
    e passar ou não os valores dos atributos
    obs: para o construtor ou os metodos criados precisamos no mínimo
passar o self para
    indicar que estamos manipalando o objeto"""
    def __init__(self, modelo, ano):
        self.modelo = modelo
        self.ano = ano
        self.km = 0

    """ metodo da classe criada:
    para criarmos um método é similar a criar uma função em python
    ou seja usamos o def com o nome do metodo.
    no caso em especifico utilizamos:
    def descricao(self): -> não passamos nem um parametro pq esse
metodo so retorna um print
    da descrição do carro """
    def descricao(self):
        print(f'Dados do Veiculo')
        print(f'Modelo: {self.modelo} Ano: {self.ano} km: {self.km}')

    def k(self, kilometro):
        self.km = self.km + kilometro
```

```
""" criando uma instância do objeto carro:
 passando 2 parametros o modelo do carro e o ano"""
c1 = Carro('Gol', 2015)

""" para acessarmos um metodo da instancia criada
devemos faze-lo com o ponto nome do metodo.
no caso aqui estamos acessando o print da descricao do carro"""
c1.descricao()

c1.k(200)

c1.descricao()
```

Fila

```
""" construindo uma fila do zero:
conceitos -> inicio da fila; fim da fila """

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Queue:
    def __init__(self):
        self.first = None
        self.last = None
        self._size = 0

    def push(self, elem):
        node = Node(elem)
        if self.last is None:
            self.last = node

        else:
            self.last.next = node
            self.last = node

        if self.first is None:
            self.first = node

        self._size += 1

    def __len__(self):
        return self._size

    def len(self):
        return self.__len__()

    def pop(self):
        if self.first:
            self.first = self.first.next
            self._size -= 1

    def peek(self):
        if self.first:
            return self.first.data

    def final(self):
```

```
        if self.last:
            return self.last.data

fila = Queue()
fila.push(3)
fila.push(2)
fila.push(4)
fila.push(5)

fila.pop()
print(fila.peek())
fila.pop()
print(fila.peek())
print(len(fila))
```

Pilha

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.top = None
        self._size = 0

    def push(self, elem):
        if self.top:
            aux = self.top
            self.top = Node(elem)
            self.top.next = aux
        else:
            self.top = Node(elem)
        self._size = self._size + 1

    def pop(self):
        if self.top:
            elem = self.top.data
            self.top = self.top.next
            return elem
        else:
            return None

    def peek(self):
        if self.top:
            return self.top.data
        else:
            return None


pilha = Stack()
pilha.push(2)
pilha.push(3)
pilha.push(4)
pilha.push(5)

print(pilha.peek())
print(pilha.pop())
print(pilha.peek())
```

Exercicio 6

```
""" Faça uma função que receba duas listas encadeadas previamente
ordenadas.
Gere uma terceira lista contendo os elementos das duas listas
recebidas, ordenados e sem repetições.
Não utilize o método sort() para este exercício."""

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self._size = 0

    def append(self, elem):
        if self.head:
            point = self.head
            while point.next:
                point = point.next
            point.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size = self._size + 1

    def __len__(self):
        return self._size

    def len(self):
        return self.__len__()

    def __getitem__(self, index):  # obs: as funções que tem o
anderline são funções reservadas do python
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
```

```
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def set(self, index, elem):
        self.__setitem__(index, elem)

    def __str__(self):
        if self.head:
            pointer = self.head
            while pointer:
                print(pointer.data)
                pointer = pointer.next
        else:
            raise []

    def string(self):
        return self.__str__()

    def index(self, elem):
        pointer = self.head
        cont = 0
        while pointer:
            if pointer.data == elem:
                return cont
            pointer = pointer.next
            cont += 1
        raise ValueError(f'{elem} is not list')

    """ questão 6: Faça uma função que receba duas listas encadeadas
previamente ordenadas.
     Gere uma terceira lista contendo os elementos das duas listas
recebidas, ordenados e sem repetições.
     Não utilize o método sort() para este exercício.
"""

    def ordena(self):
        aux = 0
        for i in range(self.len()):
            for j in range(self.len()):
                if self[i] < self[j]:
                    aux = self[i]
                    self[i] = self[j]
                    self[j] = aux

    def verifica(self, elem):
        for i in range(self.len()):
            if elem == self[i]:
                return True
        return False

    def juntarListas(self, lista):
        lista2 = LinkedList()

        for i in range(self.len()):
                lista2.append(self[i])

        for i in range(lista.len()):
```

```
            if not self.verifica(lista[i]):
                lista2.append(lista[i])

        lista2.ordena()
        return lista2

    """ questão 7: Dada uma lista, verifique quais são os elementos
que se repetem nela. Use laços aninhados.
"""
    def quant_repeticao(self):
        lista = []
        for i in range(self.len()):
            for j in range(i+1, self.len()):
                if self[i] == self[j]:
                    lista.append(self[i])
        return lista


    """ questão 8: Refaça a questão anterior sem utilizar laços
aninhados."""
    def repeticao(self):
        lista = []  # normal
        lista_repetida = []
        for i in range(self.len()):
            if i == 0:
                lista.append(self[i])
            else:
                if not self[i] in lista:
                    lista.append(self[i])
                else:
                    lista_repetida.append(self[i])
        return lista_repetida


    """ lista 2 de exercicios"""
    """ Faça um método chamado extend(), que recebe uma LinkedList e a
concatena à lista self,
    ou seja, acrescenta os elementos da lista recebida no final da
lista que dispara o método."""

    def extend(self, lista):
        if self.head:
            pointer = self.head
            while pointer.next:
                pointer = pointer.next
            pointer
        else:
            raise None


lista = LinkedList()
lista.append(3)
lista.append(6)
lista.append(2)

lista2 = LinkedList()
lista2.append(5)
lista2.append(7)
lista2.append(4)

lista.extend(lista2)
```

```
lista.string()


#lista.ordena()
#lista2.ordena()

#lista3 = lista.juntarListas(lista2)

#lista3.string()
```

ExercicioPilha

```
""" construindo a classe no ou ponteiro:
recebe um elemento e vai apontar pra outro elemento"""
class No:

    def __init__(self, dado):
        self.dado = dado
        self.proximo = None

class Pilha:
    """ construtor de uma pilha"""

    def __init__(self):
        """ no lugar de head a cabeça
        usamos o top para indicar o top da pilha"""
        self.topo = None
        self._size = 0

    """ metodo de inserção de elementos na pilha comumente
     chamado de push"""

    def push(self, elem):
        """ precisamos criar um no e dps apontalo para
        quem esta abaixo dele ou seja o antigo no sera o proximo
        e o novo no o primeiro do top da pilha """
        node = No(elem)
        node.proximo = self.topo
        self.topo = node
        self._size = self._size + 1

    """ metodo que remove o ultimo elemento ou seja o que esta no topo
    da pilha"""

    def pop(self):
        """ remove o elemento do topo da pilha """
        if self._size > 0:
            node = self.topo
            self.topo = self.topo.proximo
            self._size = self._size - 1
            return node
        raise IndexError("The stack is empty")

    def peek(self):
        """ retorna o topo da pilha sem remover
        verificando se a pilha esta vazia"""
        if self._size > 0:
            """ retornando o valor do top"""
            return self.topo.dado
```

```
    def len(self):
        return self._size

    """ imprime os elementos que tem na pilha do ultimo elemento
    inserido ao primeiro"""
    def string(self):
        """ cria uma variavel do tipo string sem nada dentro"""
        s = ""
        """ criamos um auxiliar para percorrrer todos os nos """
        pointer = self.topo
        """ enquanto tiver nós dentro da pilha continue"""
        while (pointer):
            """ vamos pegando cada letra que esta de dentro do no
            e concatenando na variavel r"""
            s = s + pointer.dado + ""
            """ e vamos atualizando o nó"""
            pointer = pointer.proximo
        """ no final retornara a string invertida"""
        return s

    def inverterPalavra(self, string):
        for i in range(len(string)):
            self.push(string[i])


pilha = Pilha()
pilha.inverterPalavra("computação")
print(pilha.string())


def peek(self):
    """ retorna o topo da pilha sem remover
    verificando se a pilha esta vazia"""
    if self._size > 0:
        """ retornando o valor do top"""
        return self.top.data

    def __len__(self):
        """ Retorna o tamanho da pilha"""
        return self._size

    def _str_(self):
        return self.__repr__()

""" um pilha atua de um modo um pouco diferente da lista encadeada
pois a nossa referencia sera sempre o top da lista
 observação de metodos da pilha:
 # inserir elementos na pilha
 # remover da pilha
 # observar o topo da pilha"""
```

Exercicio Lista 01

```
""" faça uma função para concatenar duas listas encadeadas """
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Linkedlist:
```

```
    def __init__(self):
        self.head = None
        self.__size = 0

    def append(self, elem):
        if self.head:
            aux = self.head
            while aux.next:
                aux = aux.next
            aux.next = Node(elem)
        else:
            self.head = Node(elem)
        self.__size += 1

    def __len__(self):
        return self.__size

    def len(self):
        return self.__len__()

    def __getitem__(self, index):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def set(self, index, elem):
        """ recebe um indice e o elemento e chamamos a função especial
__setitem__ com os parametros e ela fara a operação indicada """
        self.__setitem__(index, elem)

    """ função que procura um elemento na lista e retorna a posição do
mesmo caso ele exista"""
    def index(self, elem):
```

```
        pointer = self.head
        pos = 0
        while pointer:
            if pointer.data == elem:
                return pos
            pointer = pointer.next
            pos += 1
        raise ValueError(f"{elem} is not list")

    def imprimir(self):
        aux = self.head
        if aux:
            while aux:
                print(aux.data)
                aux = aux.next

    """Exercicio 1:  função que concatena duas listas encadeadas """
    def concatena(self, inicio):
        """ fazemos a mesma para percorrer a primeira lista criamos
uma
        variavel auxiliar"""
        aux = self.head
        if aux:
            """ percorremos a lista ate não ter mais elementos"""
            while aux.next:
                aux = aux.next
            """ quando auxilar estiver apontando pra nada ai podemos
apontar
            para cabeça da outra lista """
            aux.next = inicio



    """ Exercicio 2: Faça uma função que receba uma lista encadeada e
a divida em duas listas, cada uma com metade dos elementos.
     Caso haja uma quantidade ímpar de elementos, a primeira lista
retornada deve conter um elemento a mais """

    def divide_lista(self, lista2, lista3): # recebe um lista com
elementos e duas sem elementos
        aux = self.head     # testamos se a cabeça da lista tem
elementos
        if aux:
            """ se a o tamanho da lista que tem elementos for impar
fazemos esse bloco de codigo"""
            if self.len() % 2 == 1:
                """ adiciona um elemento a mais na primeira lista"""
                for i in range((self.len()//2)+1):
                    """ adiciona na lista2"""
                    lista2.append(aux.data)
                    """ e vai para o proximo no """
                    aux = aux.next
                """ vai percorrer o número inteiro da divisão da lista
por 2 """
                for i in range(self.len()//2):
                    """ o codigo vai acontecer de modo analogo ao de
cima"""
                    lista3.append(aux.data)
                    aux = aux.next
```

```
                    """ se for par fazemos esse bloco de codigos """
            else:
                """ divide a lista em 2 e divide os elementos para as
2 listas"""

                """ pega a metade da lista e passa para a primeira
lista"""
                for i in range(self.len()//2):
                    lista3.append(aux.data)
                    aux = aux.next
                """ pega a outra metade da lista e passa para a outra
lista"""
                for i in range(self.len()//2):
                    lista3.append(aux.data)
                    aux = aux.next



    """Questão 3: Dadas duas listas, verifique se ambas possuem o
mesmo tamanho. Em caso positivo,
     gere uma terceira lista contendo os elementos das duas listas
recebidas intercalados."""
    def listas_intercaladas(self, lista):
        """ verifica se as listas tem o mesmo tamanho"""
        if self.len() == lista.len():
            """ se sim criamos 2 auxiliares que vão percorrer as duas
listas encadeadas
            e adicionando a lista vazia criada"""
            aux = self.head
            aux2 = lista.head
            lista2 = Linkedlist()

            """ percorre as duas listas encadeadas com os dois laços
            cada laço vai andar cada uma das listas ate chegar no
final delas
            e ir adicionando"""
            while aux:
                lista2.append(aux.data)
                aux = aux.next
            while aux2:
                lista2.append(aux2.data)
                aux2 = aux2.next

            """ no final imprimimos a lista que foi criada e
adicionada os elementos
            das outras duas listas"""
            lista2.imprimir()



    """ Questão 4: Faça uma função que receba uma lista e um valor.
    Retorne a primeira posição em que este valor aparece na lista."""
    def procura_valor(self, value):
        """ criamos uma auxiliar que tem o a cabeça como primeiro
elemento
        sempre temos que começar a percorrer pela cabeça """
        aux = self.head
        """ verificamos se dentro do auxiliar que foi criado a lista
tem elementos"""
        if aux:
            """ criamos uma variavel para ir contando cada no que
```

```
percorremos
            ate achar o elemento procurado """
            indice = 0
            while aux:
                """ o if compara se o elemento do no atual e igual
                a o valor procurado """
                if aux.data == value:
                    """ se for retorna o indice do elemento"""
                    return indice
                """ vamos atualizando o no para o no seguinte"""
                aux = aux.next
                """ eh acrescentando mais um ao indice"""
                indice += 1
        else:
            return None


    """ Quetão 5: Considere uma lista encadeada cujos elementos estão
dispostos em ordem crescente.
    Faça uma função que receba esta lista e um valor a ser buscado.
    Informe a posição em que o valor apareceu. Para tanto, execute
busca binária."""

    def crescente(self):
        for i in range(len(self)-1):
            if self[i] > self[i+1]:
                return False
        return True

    def __crescente(self):
        for i in range(len(self) - 1):
            if self[i] > self[i + 1]:
                return False
        return True

    def __buscaBinariaInterna(self, valor, ini, fim):
        if ini > fim:
            return None
        else:
            meio = (ini + fim) // 2
            if self[meio] == valor:
                return meio
            elif valor < self[meio]:
                return self.__buscaBinariaInterna(valor, ini, meio -
1)
            else:
                return self.__buscaBinariaInterna(valor, meio + 1,
fim)

    def buscaBinaria(self, valor):
        if self.__crescente():
            return self.__buscaBinariaInterna(valor, 0, len(self) - 1)
        else:
            # deveríamos construir e lançar uma exceção com raise
            return None

    """ questão 1 de um jeito: função que concatena duas listas
encadeadas """
    def concatenar(self, lista):
        aux = self
        for i in range(len(lista)):
```

```
            aux.append(lista[i])
        aux.imprimir()

    """ questão 6: Faça uma função que receba duas listas encadeadas
previamente ordenadas.
    Gere uma terceira lista contendo os elementos das duas listas
recebidas, ordenados e sem repetições.
     Não utilize o método sort() para este exercício."""

    def ordenaLista(self):
        if self:
            aux = 0
            for i in range(len(self)-1):
                for j in range(len(self)-1):
                    if self[i] > self[j]:
                        aux = self[j]
                        self[j] = self[i]
                        self[i] = aux


#def gerarLista(self, lista):



lista = Linkedlist()
lista2 = Linkedlist()

lista.append(3)
lista.append(2)
lista.append(1)
lista.append(5)
lista.append(4)

lista.ordenaLista()
lista.imprimir()

# lista.concatenar(lista2)
# print(lista.buscaBinaria(2))
```

```
""" faça uma função para concatenar duas listas encadeadas """
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Linkedlist:
    def __init__(self):
        self.head = None
        self.__size = 0

    def append(self, elem):
        if self.head:
            aux = self.head
            while aux.next:
                aux = aux.next
            aux.next = Node(elem)
        else:
            self.head = Node(elem)
        self.__size += 1
```

```
    def __len__(self):
        return self.__size

    def len(self):
        return self.__len__()

    def __getitem__(self, index):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def set(self, index, elem):
        """ recebe um indice e o elemento e chamamos a função especial
__setitem__ com os parametros e ela fara a operação indicada """
        self.__setitem__(index, elem)

    """ função que procura um elemento na lista e retorna a posição do
mesmo caso ele exista"""
    def index(self, elem):
        pointer = self.head
        pos = 0
        while pointer:
            if pointer.data == elem:
                return pos
            pointer = pointer.next
            pos += 1
        raise ValueError(f"{elem} is not list")

    def imprimir(self):
        aux = self.head
        if aux:
            while aux:
                print(aux.data)
```

```
                aux = aux.next

    """Exercicio 1:  função que concatena duas listas encadeadas """
    def concatena(self, inicio):
        """ fazemos a mesma para percorrer a primeira lista criamos
uma
        variavel auxiliar"""
        aux = self.head
        if aux:
            """ percorremos a lista ate não ter mais elementos"""
            while aux.next:
                aux = aux.next
            """ quando auxilar estiver apontando pra nada ai podemos
apontar
            para cabeça da outra lista """
            aux.next = inicio



    """ Exercicio 2: Faça uma função que receba uma lista encadeada e
a divida em duas listas, cada uma com metade dos elementos.
     Caso haja uma quantidade ímpar de elementos, a primeira lista
retornada deve conter um elemento a mais """

    def divide_lista(self, lista2, lista3): # recebe um lista com
elementos e duas sem elementos
        aux = self.head     # testamos se a cabeça da lista tem
elementos
        if aux:
            """ se a o tamanho da lista que tem elementos for impar
fazemos esse bloco de codigo"""
            if self.len() % 2 == 1:
                """ adiciona um elemento a mais na primeira lista"""
                for i in range((self.len()//2)+1):
                    """ adiciona na lista2"""
                    lista2.append(aux.data)
                    """ e vai para o proximo no """
                    aux = aux.next
                """ vai percorrer o número inteiro da divisão da lista
por 2 """
                for i in range(self.len()//2):
                    """ o codigo vai acontecer de modo analogo ao de
cima"""
                    lista3.append(aux.data)
                    aux = aux.next


                    """ se for par fazemos esse bloco de codigos """
            else:
                """ divide a lista em 2 e divide os elementos para as
2 listas"""

                """ pega a metade da lista e passa para a primeira
lista"""
                for i in range(self.len()//2):
                    lista3.append(aux.data)
                    aux = aux.next
                """ pega a outra metade da lista e passa para a outra
lista"""
                for i in range(self.len()//2):
                    lista3.append(aux.data)
```

```
                    aux = aux.next



    """Questão 3: Dadas duas listas, verifique se ambas possuem o
mesmo tamanho. Em caso positivo,
     gere uma terceira lista contendo os elementos das duas listas
recebidas intercalados."""
    def listas_intercaladas(self, lista):
        """ verifica se as listas tem o mesmo tamanho"""
        if self.len() == lista.len():
            """ se sim criamos 2 auxiliares que vão percorrer as duas
listas encadeadas
            e adicionando a lista vazia criada"""
            aux = self.head
            aux2 = lista.head
            lista2 = Linkedlist()

            """ percorre as duas listas encadeadas com os dois laços
            cada laço vai andar cada uma das listas ate chegar no
final delas
            e ir adicionando"""
            while aux:
                lista2.append(aux.data)
                aux = aux.next
            while aux2:
                lista2.append(aux2.data)
                aux2 = aux2.next

            """ no final imprimimos a lista que foi criada e
adicionada os elementos
            das outras duas listas"""
            lista2.imprimir()



    """ Questão 4: Faça uma função que receba uma lista e um valor.
    Retorne a primeira posição em que este valor aparece na lista."""
    def procura_valor(self, value):
        """ criamos uma auxiliar que tem o a cabeça como primeiro
elemento
        sempre temos que começar a percorrer pela cabeça """
        aux = self.head
        """ verificamos se dentro do auxiliar que foi criado a lista
tem elementos"""
        if aux:
            """ criamos uma variavel para ir contando cada no que
percorremos
            ate achar o elemento procurado """
            indice = 0
            while aux:
                """ o if compara se o elemento do no atual e igual
                a o valor procurado """
                if aux.data == value:
                    """ se for retorna o indice do elemento"""
                    return indice
                """ vamos atualizando o no para o no seguinte"""
                aux = aux.next
                """ eh acrescentando mais um ao indice"""
                indice += 1
        else:
```

```
            return None


    """ Quetão 5: Considere uma lista encadeada cujos elementos estão
dispostos em ordem crescente.
    Faça uma função que receba esta lista e um valor a ser buscado.
    Informe a posição em que o valor apareceu. Para tanto, execute
busca binária."""

    def crescente(self):
        for i in range(len(self)-1):
            if self[i] > self[i+1]:
                return False
        return True

    def __crescente(self):
        for i in range(len(self) - 1):
            if self[i] > self[i + 1]:
                return False
        return True

    def __buscaBinariaInterna(self, valor, ini, fim):
        if ini > fim:
            return None
        else:
            meio = (ini + fim) // 2
            if self[meio] == valor:
                return meio
            elif valor < self[meio]:
                return self.__buscaBinariaInterna(valor, ini, meio -
1)
            else:
                return self.__buscaBinariaInterna(valor, meio + 1,
fim)

    def buscaBinaria(self, valor):
        if self.__crescente():
            return self.__buscaBinariaInterna(valor, 0, len(self) - 1)
        else:
            # deveríamos construir e lançar uma exceção com raise
            return None

    """ questão 1 de um jeito: função que concatena duas listas
encadeadas """
    def concatenar(self, lista):
        aux = self
        for i in range(len(lista)):
            aux.append(lista[i])
        aux.imprimir()

    """ questão 6: Faça uma função que receba duas listas encadeadas
previamente ordenadas.
    Gere uma terceira lista contendo os elementos das duas listas
recebidas, ordenados e sem repetições.
     Não utilize o método sort() para este exercício."""

    def ordenaLista(self):
        if self:
            aux = 0
            for i in range(len(self)-1):
                for j in range(len(self)-1):
```

```
                    if self[i] > self[j]:
                        aux = self[j]
                        self[j] = self[i]
                        self[i] = aux


#def gerarLista(self, lista):



lista = Linkedlist()
lista2 = Linkedlist()

lista.append(3)
lista.append(2)
lista.append(1)
lista.append(5)
lista.append(4)

lista.ordenaLista()
lista.imprimir()

# lista.concatenar(lista2)
# print(lista.buscaBinaria(2))
```

Filas

```
"""Apresente uma codificação em Python que simule a fila de um
atendimento de um banco.
 A codificação deve realizar a disponibilização de ficha de
atendimento, e realizar o atendimento das pessoas presentes na fila.
Utilize o conceito Fila para representar a fila bancária. Os dados das
pessoas da fila devem possuir nome, CPF e número da conta bancária.
"""

class Node:
   def __init__(self, cpf, conta):
       self.cpf = cpf
       self.conta = conta
       self.next = None


class Queue:
   def __init__(self):
       self.first = None
       self.last = None
       self._size = 0

   def push(self, cpf, conta):
       node = Node(cpf, conta)
       if self.last is None:
           self.last = node

       else:
           self.last.next = node
           self.last = node

       if self.first is None:
           self.first = node
```

```
        self._size = self._size + 1


    def pop(self):
        if self.first is not None:
            # elem = self.first.data
            self.first = self.first.next
            self._size = self._size - 1
           #  return elem
        else:
            raise IndexError("The queue is empty")

    def peek(self):
        if self._size > 0:
            elem = self.first
            return elem
        raise IndexError("the queue is empty")

    def __len__(self):
        return self._size

    def len(self):
        return self.__len__()

    def __repr__(self):
        if self._size > 0:
            pointer = self.first
            while(pointer):
                print("Dados da Conta")
                print(f'Cpf:{pointer.cpf}   Numero da
Conta:{pointer.conta}\n\n')
                pointer = pointer.next
        else:
            raise IndexError("the queue is empty")

    def __str__(self):
        self.__repr__()

    def string(self):
        print(self.__str__())



fila = Queue()
op = 's'

while op in 'Ss':
    cpf = str(input('informe o cpf: '))
    conta = int(input('informe o número da conta: '))
    fila.push(cpf, conta)
    op = str(input('insirir nova pessoa na fila: [S/N]'))

opcao = str(input('deseja imprimir os dados das pessoas da fila?\n'))
if opcao in 'Ss':
    print(fila.__str__())

op = 's'
while op in 'Ss':
    op = str(input('deseja remover os dados da primeira pessoa da fila
atualmente: [S/N]'))
    if op in 'Ss' and len(fila) > 0:
```

```
        fila.pop()
    elif not len(fila) > 0:
        op = 'n'
        print('Não tem elementos na fila de dados!!')
    else:
        print('Saindo do progama...\n')

opcao = str(input('deseja imprimir a fila dos dados das pessoas depois
da remoção: [S/N]\n'))

if opcao in 'Ss' and len(fila) > 0:
    fila.string()

elif not len(fila) > 0:
    print('Não tem elementos na fila de dados!!')

else:
    print('Progama Encerrado!!!')
```

Jogo da velha

```
def menu():
    continuar = 1
    while continuar:
        continuar = int(input("0. Sair \n" +
                              "1. Jogar novamente\n"))
        if continuar == 1:
            game()
        else:
            print("Saindo...")


def game():
    jogada = 0

    while ganhou() == 0 and jogada < 9:
        print("\nJogador ", jogada % 2 + 1)
        exibe()
        linha = int(input("\nLinha :"))
        coluna = int(input("Coluna:"))

        if board[linha - 1][coluna - 1] == 0:
            if (jogada % 2 + 1) == 1:
                board[linha - 1][coluna - 1] = 1
            else:
                board[linha - 1][coluna - 1] = -1
        else:
            print("Nao esta vazio")
            jogada -= 1

        if ganhou():
            print("Jogador ", jogada % 2 + 1, " ganhou apos ", jogada
+ 1, " rodadas")

        jogada += 1
        print("Empate! jogador1 empatou com jogador 2...")


def ganhou():
    # checando linhas
```

```
    for i in range(3):
        soma = board[i][0] + board[i][1] + board[i][2]
        if soma == 3 or soma == -3:
            return 1

    # checando colunas
    for i in range(3):
        soma = board[0][i] + board[1][i] + board[2][i]
        if soma == 3 or soma == -3:
            return 1

    # checando diagonais
    diagonal1 = board[0][0] + board[1][1] + board[2][2]
    diagonal2 = board[0][2] + board[1][1] + board[2][0]
    if diagonal1 == 3 or diagonal1 == -3 or diagonal2 == 3 or
diagonal2 == -3:
        return 1

    return 0



def exibe():
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                print(" _ ", end=' ')
            elif board[i][j] == 1:
                print(" X ", end=' ')
            elif board[i][j] == -1:
                print(" O ", end=' ')

        print()


board = [[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0]]

menu()
```

JokenPow

```
""" jo ken pow ou pedra, papel e tesoura """
from random import randint
from time import sleep

def jogar(jog, pc):
    ganhou = 0
    if jog == 1:
        if pc == 1:
            print("OS DOIS JOGARAM PEDRA DEU EMPATE! JOGAR NOVAMENTE")
            ganhou = 0
            return ganhou
        elif pc == 2:
            print("O COMPUTADOR GANHOU!\n PC JOGOU PAPEL E JOGADOR
PEDRA")
            ganhou = pc
            return ganhou
        else:
```

```
            print(" O JOGADOR GANHOU!\n JOGADOR JOGOU PEDRA E O
COMPUTADOR TESOURA")
            ganhou = jog
            return ganhou
    elif jog == 2:
        if pc == 2:
            print(" OS DOIS JOGARAM PAPEL DEU EMPATE! JOGAR
NOVAMENTE")
            ganhou = 0
            return ganhou
        elif pc == 1:
            print(" O JOGADOR GANHOU!\n JOGADOR JOGOU PAPEL E O
COMPUTADOR PEDRA")
            ganhou = jog
            return ganhou
        else:
            print(" O COMPUTADOR GANHOU!\n JOGADOR JOGOU PAPEL E O
COMPUTADOR TESOURA")
            ganhou = pc
            return ganhou
    else:
        if pc == 3:
            print(" OS DOIS JOGARAM TESOURA DEU EMPATE! JOGAR
NOVAMENTE")
            ganhou = 0
            return ganhou
        elif pc == 1:
            print(" O COMPUTADOR GANHOU!\n COMPUTADOR JOGOU PEDRA E O
JOGADOR TESOURA")
            ganhou = pc
            return ganhou
        else:
            print(" O JOGADOR GANHOU!\n JOGADOR JOGOU TESOURA E O
COMPUTADOR PAPEL")
            ganhou = jog
            return ganhou



print("="*10, " VAMOS JOGAR JO KEN POW ", "="*10)
print("="*46)
sleep(1.5)
print("GANHA QUEM VENCER AS 5 PRIMEIRA PARTIDAS...")
print("="*46)
contador_jog = 0
contador_pc = 0
while contador_jog < 3 and contador_pc < 3:
    sleep(1.5)
    jog = int(input('ESCOLHA UMA OPÇÃO:\n[ 1 ] PEDRA\n[ 2 ] PAPEL\n[ 3
] TESOURA\n OPÇÃO:'))
    if jog > 3:
        print("OPÇÃO INVÁLIDA! TENTE NOVEMENTE...")
    else:
        print("JO!!!")
        sleep(1)
        print("KEN!!!")
        sleep(1)
        print("POW!!!")
        sleep(1)
        pc = randint(1, 3)
        res = jogar(jog, pc)
```

```
        if res == jog:
            contador_jog += 1
        elif res == pc:
            contador_pc += 1
        else:
            pass
        print()
        print("="*46)
sleep(1)
print(f'O PLACAR FOI {contador_jog} VITORIAS PARA JOGADOR E
{contador_pc} PARA O PC...')
```

LinkdList

```
""" A classe Node vai ser o nó da nossa lista onde faremos as ligações
com os elementos
criando uma classe Node do ingles nó que deve ter dois atributos: data
que do ingles e um dado e next que e  proximo ou seja
uma variavel dado que vai receber um elemento e apontar para proximo
elemento da lista que e a variavel next que inicilamente e None
(não existe) """
class Node:
    """ construtor que recebe que recebe um dado no parametro """
    def __init__(self, data):
        self.data = data
        self.next = None

""" com a classe Node ja criada podemos criar a classe Linklist que
significa lista encadeada """
class LinkedList:

    """ construtor da classe LinkedList que vai iniciar com dois
atributos: head ( a cabeça ) que vai receber o primeiro elemento da
lista
     ou seja a variavel head sera a referencia para andarmos por toda
a lista eh a variavel size que retorna o tamanho da lista que
inicilmnte e
     zero mais vamos modificando ela de acordo com a inserção dos
elementos """
    def __init__(self):
        self.head = None
        self._size = 0  # obs: o __variave faz o encapsulamento do
atributo para que estiver de fora não possa acessala diretamente

    """ a função sera a responsavel por inserir os elementos na lista.
ela funciona de modo analogo a de uma lista estatica em python onde:
    por exemplo:
    lista = [] -> lista estatica em python e que esta vazia
    lista.append(elemento) -> adiciona um elemento a lista

    ou seja faremos algo parecido so que teremos que implatar nosso
proprio append onde receberemos o elemento e faremos as devidas
operações """
    def append(self, elem):
        if self.head:
            """ se a lista tiver elemento então criamos um ponteiro
            que vai apontar para o mesmo endereco que a cabeça do nó
            que no caso eh self.head
            obs: temos que criar um variavel auxiliar para andarmos
por ela por que não podemos andar pelo self.head ( cabeça ) pois
            ela gurdara o primeito elemento inserido ou seja a
```

```
referencia pra irmos andando nessa lista encadeada """

            pointer = self.head
            """ a variavel auxiliar recebe self.head, com isso podemos
andar pela variavel auxiliar
            pointer sem modicar a lista ou seja a variavel pointer tem
os mesmos atributos do objeto Node
            que eh: data um dado e um varivel para o proximo endereco

            agora andaremos pelo pointer ate acharmos um valor igual a
None ou seja vazio.
            Enquanto ponteiro.next tiver elementos o laço deve
continuar a sua execução
            e so vai parar quando encontrar um valor = None ou seja
espaço vazio e sempre
            vai atualizando """
            while pointer.next:
                pointer = pointer.next
            """ depois que o ponteiro estiver apontado para vazio o
ponteiro.next ou seja o proximo elemento sera
            um novo nó que tem uma nova cabeça e um novo next vazio ou
seja a cada inserção vamos criando um novo nó
             que recebe um novo elemento na cabeça e tem um endereço
next que aponta para o vazio """
            pointer.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size += 1
        """ caso o head a cabeça da lista esteja vazia ai então
criamos uma instancia da classe Node
        e ela será a referencia para andarmos na lista encadeada """


    # função especial do objeto que retorna o tamanho do objeto
    """ não eh indicado usar essas funções para retorno e sim criar
funções
    que acessem indiretamente essa funções e retornem os valores """
    def __len__(self):
        """ retorna o tamanho do objeto"""
        return self._size

    """ funçõa que acessa indiretamente uma função interna do objeto
__len__ e retorna o valor """
    def len(self):
        return self.__len__()

    # outra função reservada para acessar um indice de um objeto e
retorna o objeto
    def __getitem__(self, index):  # obs: as funções que tem o
anderline são funções reservadas do python
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
```

```
            raise IndexError("list index out of range")

    # função que retorna o elemento pesquisado na lista
    def get(self, index):
        """ chamamos a função __getitem__ e passamos o indice e ela
retorna o elemento caso ele exista se não uma mensagem de erro """
        return self.__getitem__(index)

    # função reservada para modificar um elemento da lista passando um
indece
    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    # analogamente ao get temos a função que representa o set que
recebe o indice e o elemento para modificar na lista
    def set(self, index, elem):
        """ recebe um indice e o elemento e chamamos a função especial
__setitem__ com os parametros e ela fara a operação indicada """
        self.__setitem__(index, elem)

    def index(self, elem):
        pointer = self.head
        pos = 0
        while pointer:
            if pointer.data == elem:
                return pos
            pointer = pointer.next
            pos += 1
        raise ValueError(f"{elem} is not list")


""" criamos uma lista e depois inserimos elementos nela"""
lista = LinkedList()
lista.append(5)
lista.append(3)
lista.append(6)
lista.append(2)
lista.append(4)

print(lista[3])
""" imprimindo na tela o elemento do indece 3"""
#print(lista.get(3))

""" modificando o elemento de indice 3 para valor 1"""
#lista.set(3, 1)

""" imprimindo novamente o elemento de indice 3 com um valor diferente
do de antes"""
```

```
#print(lista.get(1))

""" procurando elemento na lista encadeada"""
#print(lista.index(1))
```

Lista de exercício 2

```
class Node:
    def __init__(self, dado):
        self.dado = dado
        self.prox = None

class LinkedList:
    def __init__(self):
        self.head = None
        self._size = 0


    def append(self, elem):
        if self.head:
            aux = self.head
            while aux.prox:
                aux = aux.prox
            aux.prox = Node(elem)
        else:
            self.head = Node(elem)
        self._size += 1


    def imprimir(self):
        if self.head:
            aux = self.head
            while aux:
                print(aux.dado)
                aux = aux.prox

    """questão 1: Faça um método chamado extend(), que recebe uma
LinkedList e a concatena à lista self,
     ou seja, acrescenta os elementos da lista recebida no final da
lista que dispara o método.
"""
    def extend(self, lista):
        if self.head:
            aux = self.head
            while aux:
                aux = aux.prox
            aux.prox = lista.head
            del lista.head
            lista.head = None
        return lista

    """questão 2: Implemente o método insert(),
     que pode inserir um novo elemento em qualquer posição da
lista."""
    def insert(self, pos, elem):
        if self.head and self._size > pos:
            aux = self.head
            cont = 0
            while aux:
                if cont == pos:
```

```
                    aux.dado = elem
                cont += 1
                aux = aux.prox

        else:
            raise IndexError("Não tem elementos na lista")

    """ questão 3: Implemente o método pop(),
    que remove o elemento na posição indicada na lista.
    Caso não haja indicação de posição, remove a última posição.
"""

    def pop(self):
        if self.head:
            aux = self.head
            aux2 = aux
            while aux:
                aux2 = aux
                aux = aux.prox
            aux = aux2




lista = LinkedList()
lista.append(2)
lista.append(3)
lista.append(5)

lista.imprimir()
print()
lista.pop()
print()
lista.imprimir()
#lista.insert(1, 4)
#lista.imprimir()

#lista2 = LinkedList()
#lista.append(4)
#lista.append(5)

#lista.extend(lista2)
#lista.imprimir()
#print(lista2.head)
#lista2.imprimir()
```

Lista encadeada

```
""" faça uma lista encadeada com varios metodos: inserir, remover,
retorna um valor, mudar um valor ...."""

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```
        self._size = 0

    """ essa função inseri no final da lista encadeada"""
    def append(self, elem):
        if self.head:
            point = self.head
            while point.next:
                point = point.next
            point.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size = self._size + 1


    """ essa função inseri no inicio da lista encadeada """
    def append_inicio(self, value):
        if self.head:
            aux = self.head
            self.head = Node(value)
            self.head.next = aux


    def __len__(self):
        return self._size

    def len(self):
        return self.__len__()

    def __getitem__(self, index):  # obs: as funções que tem o
anderline são funções reservadas do python
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")
```

```
    def set(self, index, elem):
        self.__setitem__(index, elem)

    def __str__(self):
        if self.head:
            pointer = self.head
            while pointer:
                print(pointer.data)
                pointer = pointer.next
        else:
            raise []

    def string(self):
        self.__str__()

    def index(self, elem):
        pointer = self.head
        cont = 0
        while pointer:
            if pointer.data == elem:
                return cont
            pointer = pointer.next
            cont += 1
        raise ValueError(f'{elem} is not list')



lista = LinkedList()
lista.append(3)
lista.append(4)
lista.append(6)

lista.append_inicio(4)

lista.string()
```

Lista Exercicio 2

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self._size = 0

    def append(self, elem):
        if self.head:
            point = self.head
            while point.next:
                point = point.next
            point.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size = self._size + 1

    def __len__(self):
```

```
        return self._size

    def len(self):
        return self.__len__()

    def __getitem__(self, index):  # obs: as funções que tem o
anderline são funções reservadas do python
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            return pointer.data
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def set(self, index, elem):
        self.__setitem__(index, elem)

    def __str__(self):
        if self.head:
            pointer = self.head
            while pointer:
                print(pointer.data)
                pointer = pointer.next
        else:
            raise []

    def string(self):
        return self.__str__()

    def index(self, elem):
        pointer = self.head
        cont = 0
        while pointer:
            if pointer.data == elem:
                return cont
            pointer = pointer.next
```

```
            cont += 1
        raise ValueError(f'{elem} is not list')


    """ Faça um método chamado extend(), que recebe uma LinkedList e a
concatena à lista self, ou seja,
     acrescenta os elementos da lista recebida no final da lista que
dispara o método."""

    def extend(self, lista):
        if self.head:
            aux = self.head
            while aux.next:
                aux = aux.next
            aux.next = lista
            del lista
            lista = None

lista = LinkedList()
lista2 = LinkedList()

lista.append(3)
lista.append(5)
lista.append(4)

lista2.append(2)
lista2.append(7)
lista2.append(6)
```

Pilhas

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    """ construtor de uma pilha"""
    def __init__(self):
        """ no lugar de head a cabeça
        usamos o top para indicar o top da pilha"""
        self.top = None
        self._size = 0

    """ metodo de inserção de elementos na pilha comumente
     chamado de push"""
    def push(self, elem):
        node = Node(elem)
        node.next = self.top
        self.top = node
        self._size = self._size + 1

    """ metodo que remove o ultimo elemento ou seja o que esta no topo
    da pilha"""
    def pop(self):
        """ remove o elemento do topo da pilha """
        if self._size > 0:
            node = self.top
            self.top = self.top.next
            self._size = self._size - 1
```

```
            return node
        raise IndexError("The stack is empty")

    def peek(self):
        """ retorna o topo da pilha sem remover
        verificando se a pilha esta vazia"""
        if self._size > 0:
            return self.top.data

    def __len__(self):
        """ Retorna o tamanho da pilha"""
        return self._size

    def len(self):
        return self.__len__()

    """ imprime os elementos que tem na pilha do ultimo elemento
    inserido ao primeiro"""
    def __repr__(self):
        r = ""
        pointer = self.top
        while (pointer):
            r = r + pointer.data + ""
            pointer = pointer.next
        return r

    def string(self):
        print(self.__repr__())


    def inverteString(self, string):
        for i in range(len(string)):
            self.push(string[i])

pilha = Stack()
pilha.inverteString("uepb")
print(pilha.string())
```

Prova 3

```
""" """
class Node:
    def __init__(self, num):
        self.num = num
        self.next = None

class Vetor:
    def __init__(self):
        self.head = None
        self._size = 0

    def append(self, elem):
        if self.head:
            point = self.head
            while point.next:
                point = point.next
            point.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size = self._size + 1
```

```
    def len(self):
        return self._size

    def ler_vetor(self, tam):
        for i in range(tam):
            valor = int(input('digite algo: '))
            self.append(valor)

    def imprimir(self):
        aux = self.head
        for i in range(self.len()):
            print(aux.num)
            aux = aux.next

    def conta_negativos(self):
        cont = 0
        aux = self.head
        for i in range(self.len()):
            if aux.num < 0:
                cont = cont + 1
            aux = aux.next
        return cont


vetor = Vetor()
vetor.ler_vetor(6)
vetor.imprimir()
print(vetor.conta_negativos())
```

Prova questão 1

```
class Node:
    def __init__(self, num):
        self.num = num
        self.next = None

class Vetor:
    def __init__(self):
        self.head = None
        self._size = 0

    def append(self, elem):
        if self.head:
            point = self.head
            while point.next:
                point = point.next
            point.next = Node(elem)
        else:
            self.head = Node(elem)
        self._size = self._size + 1

    def __getitem__(self, index):  # obs: as funções que tem o
anderline são funções reservadas do python
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
```

```
                raise IndexError("list index out of range")
        if pointer:
            return pointer.num
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def get(self, index):
        return self.__getitem__(index)

    def __setitem__(self, index, elem):
        pointer = self.head
        for i in range(index):
            if pointer:
                pointer = pointer.next
            else:
                """ palavra reservada de python para uso em objetos"""
                raise IndexError("list index out of range")
        if pointer:
            """ aqui a uma mudança por que não queremos o retorno
            do elemento da lista e sim modificalo"""
            pointer.data = elem
        else:
            """ gerando o erro novamente"""
            raise IndexError("list index out of range")

    def set(self, index, elem):
        self.__setitem__(index, elem)

    def len(self):
        return self._size

    def ler_vetor(self, tam):
        for i in range(tam):
            valor = int(input('digite algo: '))
            self.append(valor)

    def imprimir(self):
        aux = self.head
        for i in range(self.len()):
            print(aux.num)
            aux = aux.next

    def ordenar(self):
        for i in range(self._size):
            for j in range(self._size):
                if self[i] < self[j]:
                    aux = self[i]
                    self[i] = self[j]
                    self[j] = aux
        self.imprimir()

    def ordenado(self):
        for i in range(self._size):
            for j in range(self._size):
                if self[i] > self[i+1]:
                    return True
        return False
```

```
    """def conta_negativos(self):
        cont = 0
        aux = self.head
        for i in range(self.len()):
            if aux.num < 0:
                cont = cont + 1
            aux = aux.next
        return cont"""


vetor = Vetor()
vetor.ler_vetor(4)
print(vetor.ordenado())
vetor.ordenar()
#print(vetor.conta_negativos())
```