1) Utilizando a estrutura do tipo vetor, faça um programa que simule uma pilha, com suas funções de push e pop, assim como, a função de consultar o topo da pilha, o primeiro elemento da pilha, ou seja, a base, e listar os elementos que estão na pilha informando a ordem de saída de cada elemento.

Obs: Os códigos devem estar com comentários claros e diretos nos principais trechos.

A ideia da pilha é empilhar um elemento sobre o outro, por exemplo supondo que tenho uma pilha de livros pra retirar o livro do meio é preciso desempilhar os elementos que estão no topo depois empilhá-los novamente quando necessário. Isso quer dizer que na pilha a pessoa só tem acesso ao ultimo elemento (o topo da pilha) e os demais não temos acesso.
A pilha permite acesso a um item de dados que é sempre o último item inserido, se o ultimo item for removido o item anterior ao último inserido poderá ser acessado.

As aplicações prática sem pilha são correções aritméticas p ex 3 * (4 + 5) em comiladores se faltar ") " vai ocorrer um erro de expressão. Assim essa aplicação é um validador de expressões aritmética.
Em estrutura de dados pode utilizar pilha para percorrer uma árvore binária ou a pesquisa de um vértice de um grafo.
A aplicação mais interessante é a tela azul do windows XP quando estourava a memória (stack overflow) esse erro acontecia quando não tinha espaço na memória, quando o usuário forçava a alocação de dados na pilha da memória, o sistema operacional falhava devido ao estouro de memória.

Como na questão está solicitado: push, pop, top, e list:

inserir(dado) --> inserir um dado sempre no topo da pilha
remover() ---> remove o dado que está no topo da pilha
top() --> retorna o elemento que está no topo da pilha
listar() ---> listar todos os elementos empilhados

Obs: em nenhuma hipótese é permitido remover um dado que está no meio da pilha, porque é uma estrutura LIFO (último a entrar é o primeiro a sair)

```
import numpy as np
class Pilha:

  def __init__(self, capacidade): # o construtor é obrigatorio informar
a capacidade da pilha
    self.__capacidade = capacidade
    self.__topo = -1 # indica onde está o topo da pilha, se for -1
entao a pilha ta vazia
    self.__valores = np.empty(self.__capacidade, dtype=int) # os
valores da pilha do tipo inteiro
    # pra proxima questão é só mudar o tipo int para "String" que ler
os dados em texto para cadastrar o nome
```

```
    # dos personagens da Marvel em uma pilha

  def __pilha_vazia(self): # verificar se a pilha esta vazia
    if self.__topo == -1:
      return True # a pilha esta vazia
    else:
      return False # tem ao menos um elemeto na pilha

  def __pilha_cheia(self): # verificar se a pilha esta cheia
    if self.__topo == self.__capacidade - 1:
      return True # a pilha esta cheia
    else:
      return False # nao tem nada na pilha


  def push(self, valor): # e obrigatorio informar o dado que quer
empilhar
    if self.__pilha_cheia():
      print('nao cabe mais nada na pilha')
    else:
      self.__topo += 1 # incrementa o topo da pilha
      self.__valores[self.__topo] = valor # o valor do topo+1 recebe o
atributo valor

  def pop(self):
    if self.__pilha_vazia():
      print('nao tem nada na pilha, ela esta vazia')
    else:
      self.__topo -= 1 # nao precisa apagar o elemento e so decrementar
o topo
      # o gerenciador de lixo de memória do python desaloca da memória
automagicamente

  def top(self):
    if self.__topo != -1: # se a pilha não estiver vazia
      return self.__valores[self.__topo] # retorne o topo da pilha
    else:
      return -1 # se não entao -1 signfica que a pilha está vazia
  def listar(self):
    if self.__topo != -1: # se a pilha não estiver vazia
        for i in range(self.__topo+1):
            print(self.__valores[i])
    else:
```

```
      return -1 # se não entao -1 signfica que a pilha está vazia

if __name__ == "__main__": # funcao principal + testes
    pilha = Pilha(5) # criar um objeto pilha que cabe 5 elementos
    pilha.push(1)
    pilha.push(2)
    pilha.push(3)
    pilha.push(4)
    pilha.push(5)
    pilha.listar()
    print("\n")
    pilha.pop()
    pilha.pop()
    pilha.listar()
    print("topo da pilha: ", pilha.top())


>> Saída
      1
      2
      3
      4
      5

      1
      2
      3
      topo da pilha: 3
```

2) Utilizando registros faça uma programa que simule uma pilha, onde o usuário pode inserir personagens da Marvel, e poderá executar as funções de push e pop.

Altera o código anterior para ao invés de receber inteiro, receber uma string

```
self.__valores = np.empty(self.__capacidade, dtype=str) # os valores da pilha do tipo string. np empty forma um vetor estático com a capacidade definida pelo usuário e o tipo do dado é string
```

Depois na função principal é só fazer as mesmas operações, a diferença é incluir o nome dos personagens:

```
if __name__ == "__main__": # funcao principal + testes
```

```
    pilha = Pilha(5) # criar um objeto pilha que cabe 5 elementos
    pilha.push('Homem Aranha')
    pilha.push('Batman')
    pilha.push('Super Man')
    pilha.push('Mulher Gato')
    pilha.push('Capitão América')
    pilha.listar()
    print('\n')
    pilha.pop()
    pilha.pop()
    pilha.listar()
    print('topo da pilha:', pilha.top())

>>> Saída:
      Homem Aranha
      Batman
      Super Man
      Mulher Gato
      Capitão América

      Homem Aranha
      Batman
      Super Man
      topo da pilha: Super Man
```

Referências:

Slides das aulas teóricas do Prof. Raul Benites Paradeda (Estrutura de Dados 2020)

O código-fonte está e disponível e rodando no repositório do github disponível em:
https://github.com/abnermuxah/esturua-de-dados

Aulas teóricas do prof Jones (IA Expert Academy)
https://iaexpert.academy/courses/estrutura-de-dados-e-algoritmos-python-guia-completo/