Pilhas Sequenciais e Encadeadas

1. Faça um programa que utilize a classe **Pilha.py** estudada em sala de aula e manipule dados do tipo inteiro utilizando as seguintes opções de um *menu*:

```
Editor de Pilha v1.2
====================================
Pilha Selecionada: 3 de 10
[ ] <- topo
====================================
(e) Empilhar
(d) Desempilhar
(t) Tamanho
(o) Obter elemento do topo
(v) Teste de pilha vazia
(r) Criar nova Pilha
(n) Inverter os elementos da pilha
(z) Esvaziar a pilha
(c) Concatenar duas pilhas
(m) Escolher outra pilha
(n) Conversão dec/bin
(s) Sair
====================================
Digite sua opção: [ _ ]
```

Número de pilhas disponíveis, criadas a partir do programa

Pilha atual selecionada

Observações:

- As pilhas são inicializadas automaticamente pelo programa;
- O programa deve fazer o tratamento das mensagens de entrada e saída de dados, visando a interação com o usuário para alertar sobre o sucesso ou falha da operação. Mostre mensagens quando quiser alertar sobre uma pilha vazia ou cheia;
- A pilha deve ser atualizada automaticamente no topo do programa sempre que alguma operação do menu for executada;
- Ao desempilhar um elemento da pilha, mostre o valor que foi removido do topo.
- Permita que o usuário possa criar e manipular o número de pilhas que desejar. A primeira pilha a ser utilizada pelo usuário é a pilha **1,** considerando que o programa deve iniciar ao menos com uma pilha disponível**.** Quando a opção "escolher outra pilha" for acionada, permita que o usuário possa escolher qualquer pilha válida, dentre a quantidade existente. Se a pilha **2** for escolhida (pilha selecionada no momento), todas as operações do menu devem ser realizadas na pilha **2**, e assim por diante.

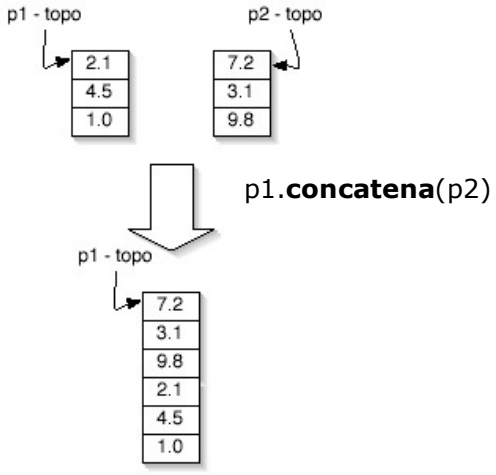
2. Adicione à questão 1 (e ao programa, via opção do menu) as seguintes operações adicionais:

```
def esvaziar( self )
    '''
    Retorna True se foi possível esvaziar a pilha ou False caso contrário
    '''
```

```
def inverte( self )
    '''
    Retorna True se foi possível inverter a pilha ou False caso contrário
    '''
```

As duas operações devem ser implementadas usando apenas as operações básicas.

3. Usando apenas as operações básicas declaradas no arquivo **`pilha.py`**, implemente um método de instância que receba uma segunda pilha p2, e transfira todos os elementos da pilha p2 para o topo da pilha responsável pela invocação da operação. A figura a seguir ilustra essa concatenação de pilhas:



Note que ao final da operação, a pilha p2 vai estar vazia e a pilha p1 conterá todos os elementos das duas pilhas. A operação deve obedecer ao seguinte protótipo:

```
def concatena( self, pilha2 )
```

Implemente, também, um método de classe que receba duas pilhas distintas e retorne uma terceira pilha com a concatenação das pilhas de entrada, sem remover os elementos das pilhas de entradas. A pilha resultante deve ter o mesmo resultado ilustrado pela figura acima. O protótipo da operação é definido por:

```
def concatenaPilhas( cls, pilha1, pilha2 )
```

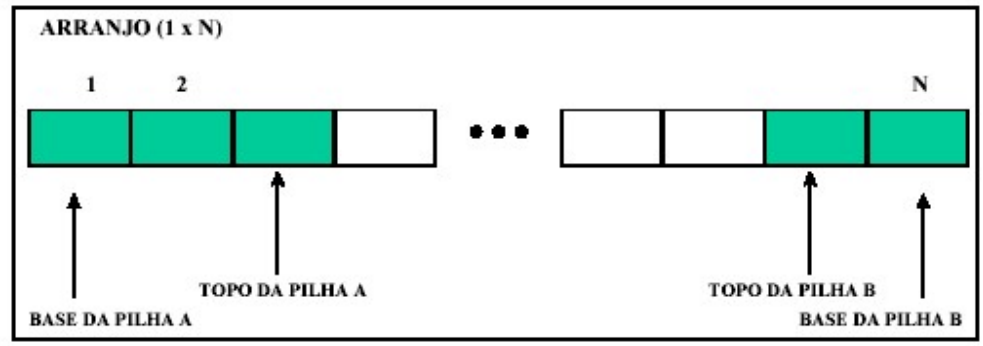
4. Implemente uma função que receba um número inteiro e retorne sua representação convertida para a base binária. Uma pilha deve ser utilizada, de forma coerente, para auxiliar a tarefa. O protótipo da função é definido por:

```
def decToBin( numero )
  '''
  Converte um número inteiro para binário
  Argumentos:
      int numero – Um número inteiro
  Exemplo de uso
      decToBin(10);  // Imprime 1010
  '''
```

5. Utilizando a mesma pilha de inteiros, acrescente uma **função** ao menu principal da questão 1 que receba uma linha de texto e a imprima de forma invertida. Analise o local ideal, dentro do projeto, para codificar essa função.

6. Ainda, na mesma aplicação, implemente uma função que use uma pilha para determinar se uma string é palíndromo, isto é, se tem a mesma leitura no sentido normal e no sentido inverso. A função deve ignorar espaços em branco, pontuação e caracteres especiais (-, *, \). Adicione essa funcionalidade ao menu da questão 1. Analise o local ideal, dentro do projeto, para codificar essa função.

7. Considere um arranjo unidimensional utilizado para implementar duas pilhas. As bases das pilhas estarão uma em cada extremidade do arranjo, conforme a figura abaixo:



O arranjo possui **N** elementos, inteiros. Cada uma das pilhas apresenta um índice de topo de pilha, os quais são nulos quando a pilha é vazia. Deste modo, faça o que se pede:

a) Crie um novo arquivo **PilhaSequencialDupla.py** e preserve as operações primitivas já discutidas em sala de aula.
b) Prepare a pilha para que ela possa ter um tamanho fixo, a ser definido como argumento no momento da instanciação
c) A pilha deve dividir automaticamente o array para que possa manipular duas pilhas internamente.
d) Adicione um método para incluir um elemento no topo da pilha B. Da mesma forma, permita que um novo elemento possa ser empilhado na pilha A. Pense na melhor maneira de implementar esse método
e) Revise os demais métodos para que possa se adequar à nova realidade. Por exemplo, o método tamanho() deve devolver o tamanho tanto da pilha A como da pilha B (não simultaneamente)
f) Adicione um procedimento que calcule a quantidade de elementos da pilha A que também aparecem na pilha B (interseção). Elementos repetidos na mesma pilha devem ser contados apenas uma vez.