FIAP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Computational Thinking

PROF. EDUARDO GONDO

If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident.

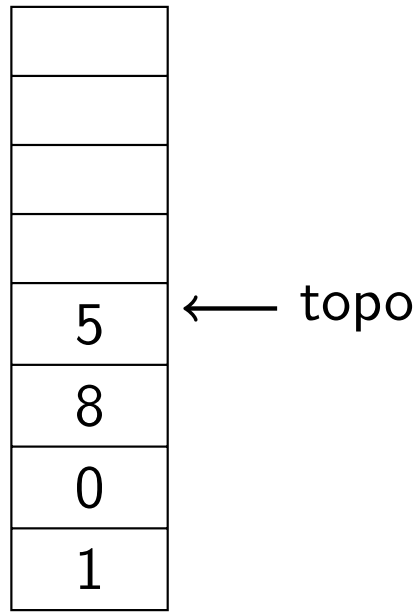
Robert Pike

Estrutura de Dados

- modo como armazenamos e manipulamos as informações dentro dos nossos algoritmos
- na prática consiste nas variáveis e nas funções que acessam essas variáveis
- ou trazendo para o paradigma orientado a objetos, consiste nos atributos e métodos de uma ou mais classes
- as estruturas de dados que apresentaremos são a **pilha** e a **fila** que são exemplos de **listas lineares**
- basicamente elas diferem no modo como as informações são inseridas e recuperadas dessas listas
- mas podemos resolver vários problemas apenas organizando como armazenamos as informações dentro dessas estruturas

Pilha (Stack) — Introdução

- LIFO - Last In First Out
- todas as inserções e remoções são feitas em uma das extremidades da lista
- no caso da pilha, denominamos essa extremidade de topo
- podemos imaginar que a pilha dentro da programação funciona igual a uma pilha de pratos do restaurante
- usaremos uma lista do Python para representar a pilha
- ao lado temos a representação gráfica da pilha



Pilha — Operações

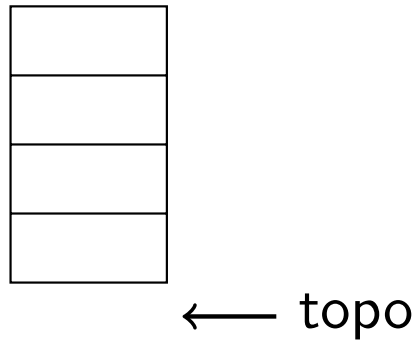
Uma pilha possui as seguintes operações (funções):

- `def isFull(pilha)`: retorna True se a pilha está cheia
- `def isEmpty(pilha)`: retorna True se a pilha está vazia
- `def put(pilha, info)`: coloca info na pilha
- `def pop()`: devolve info da pilha removendo-a da pilha
- `def peek()`: devolve a info da pilha sem removê-la

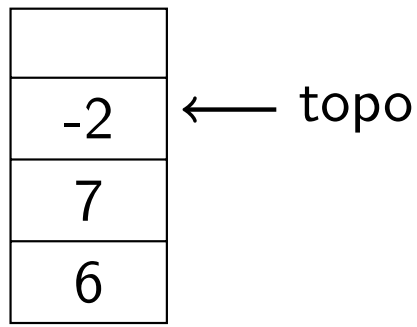
Antes de mostrarmos a implementação de uma Pilha, vamos mostrar como fica o desenho da pilha após a execução de algumas instruções:

Pilha — Teste de Mesa

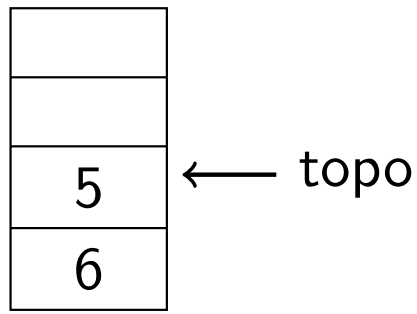
```
1   pilha = []
```




```
1   put(pilha, 6)
2   put(pilha, 7)
3   put(pilha, -2)
```

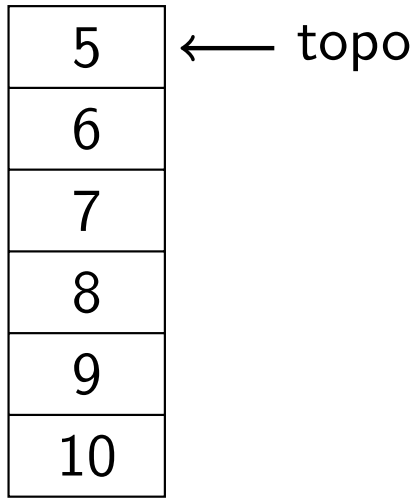



```
1   pop(pilha)
2   pop(pilha)
3   put(pilha, 5)
```



Pilha — Teste de Mesa 2

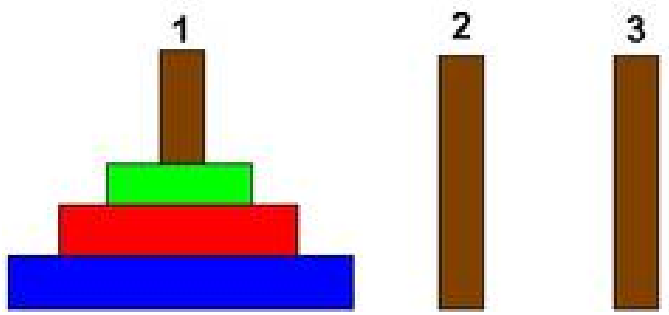
```
1  pilha = []
2  i = 10
3  while i >= 5:
4      put(pilha, i)
5      i = i - 1
```



Pilha — Torres de Hanoi

EXEMPLO 1: Torre de Hanoi é um jogo que consiste de 3 pinos representando as torres e n discos de diâmetros diferentes colocados em um dos pinos. Veja a figura abaixo representando as torres de Hanoi com 3 discos ($n = 3$). O objetivo do jogo é movimentar todos os discos do pino A para o pino C obedecendo duas regras:

- apenas um disco pode ser movimentado por vez
- um disco de diâmetro menor nunca pode ficar sobre um disco de diâmetro maior



Pilha — Sequência de símbolos

EXEMPLO 2: A maioria dos compiladores consegue perceber quando uma sequência de símbolos: (), [] e está bem formada ou não

Seguem abaixo alguns exemplos de sequências:

- `[[({}){}[{}]]]` *ok*
- `[()])` *não ok*
- `{{{)}}}` *não ok*
- `[[(])]` *não ok*
- `()()[{}]` *ok*
- `)` *não ok*

O problema consiste em você receber uma sequência de símbolos e decidir se essa sequência é bem formada ou não

Pilha — Sequência de símbolos (continuação)

- Um dos modos de resolver o problema é classificar os símbolos entre aqueles que são de abertura e os que são de fechamento
- Sempre um símbolo de abertura vai para a pilha
- Quando é lido um símbolo de fechamento, desempilha a pilha e verifica se os símbolos "casam"ou "não casam"
- Note que não deve sobrar nenhum símbolo na pilha ao fim da sequência de símbolos

Vamos executar o algoritmo acima no papel desenhando a pilha e colocando os símbolos dentro dela.

Pilha — Implementação

Vejamos abaixo um exemplo de implementação de uma pilha:

```
1  #como a lista não possui limitação, esta função sempre
       retorna False
2  def isFull(pilha):
3      return False
4
5  def isEmpty(pilha):
6     return len(pilha) == 0
7
8  def put(pilha, info):
9     pilha.append(info)
10
11 #antes de chamar as funções pop e peek, certifique-se que
       a pilha não está vazia
12 def pop(pilha):
13    return pilha.pop()
14
15 def peek(pilha):
16    pos = len(pilha) - 1
17    return pilha[pos]
```

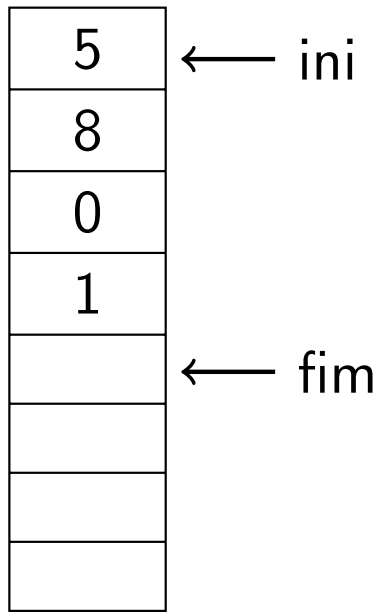
Pilha — Expressão aritmética

Exemplo 3:

- usamos a notação infixa para resolver contas: $(3 + 5) * 2$
- mas as calculadoras HP usam a notação pós-fixa ou polonesa
- $35 + 2*$
- essa notação elimina a necessidade de parênteses e colchetes para determinar a precedência dos operadores
- usaremos uma pilha para escrever um algoritmo para resolver as expressões na notação polonesa
- além de resolvê-las, elas também verificam se a expressão é correta
- vamos fazer alguns exemplos:

Fila (Queue) — Introdução

- FIFO - First In First Out
- inserções são feitas no fim da lista e remoções são feitas no início da lista
- podemos imaginar que a fila dentro da programação funciona igual a uma fila de banco ou a fila do ônibus da FIAP
- na computação temos a fila de processos e a fila de impressão
- usaremos uma lista do python para representar uma fila
- e isso facilitará e muito a implementação



Fila — Operações

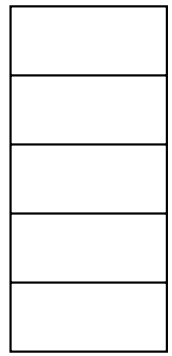
Uma fila possui as seguintes operações (funções):

- `def isEmpty(fila)`: retorna true se a fila está vazia
- `def isFull(fila)`: retorna true se a fila está cheia
- `def put(fila, info)`: coloca info na fila
- `def get(fila)`: devolve a 1ª informação da fila removendo-a da fila
- `def peek(fila)`: devolve a 1ª informação da fila sem removê-la

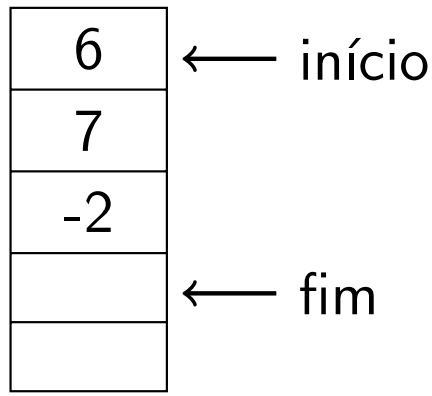
Antes de mostrarmos a implementação de uma Fila, vamos mostrar como fica o desenho dela após a execução de algumas instruções:

Fila — Teste de Mesa

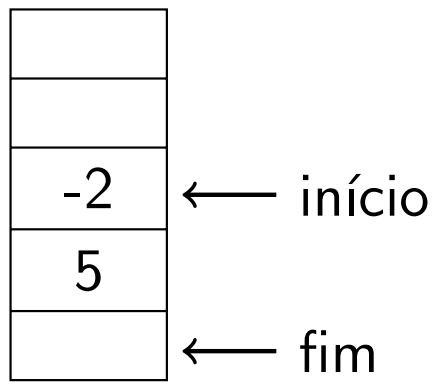
```
1   fila = []
```



```
1   put(fila, 6)
2   put(fila, 7)
3   put(fila, -2)
```

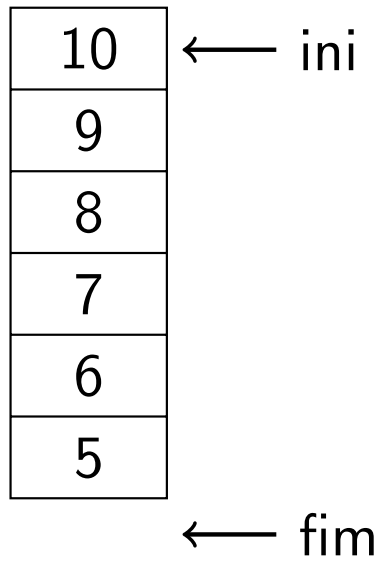



```
1   get(fila)
2   get(fila)
3   put(fila, 5);
```



Fila — Teste de Mesa 2

```
1  f = []
2  i = 10
3  while i > 4:
4      put(f, i)
5      i = i - 1
```



Fila — Implementação

Abaixo segue a implementação das funções da fila:

```
1   #como a lista não possui limitação, esta função sempre
        retorna False
2   def isFull(fila):
3       return False
4
5   def isEmpty(fila):
6      return len(fila) == 0
7
8   def put(fila, info):
9      fila.append(info)
10
11  #antes de chamar as funções get e peek, certifique-se que
        a fila não está vazia
12  def get(fila):
13     return fila.pop(0)
14
15  def peek(fila):
16     return fila[0]
```

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft