UniAcademia

Disciplina

Estrutura de Dados

prof. Jacimar Tavares
jacimar.tavares@gmail.com

Módulo 03

Estruturas de Dados Básicas

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Estruturas lineares podem ser consideradas como tendo duas extremidades.
  - Às vezes essas extremidades são chamadas de *esquerda* e *direita* ou, em alguns casos, de *frente* e *traseira*. Você também pode chamá-las de *topo* e *base*.
  - O que distingue uma estrutura linear de outra é a maneira em que itens são inseridos e removidos, em particular a extremidade onde estes inserções e remoções ocorrem.

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Listas

```
myList = list()

myList = list([1,2])

myList = [1,2]
```

Estruturas de Dados Básicas

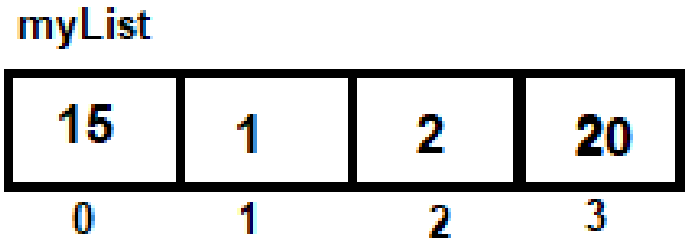
- **Estruturas Lineares**
  - Listas: Podemos inserir de duas formas, inserção sempre no final da lista, utilizando a função **append(elemento) ou,** inserção sempre na posição informada como parâmetro, utilizando a função **insert(index, elemento).**
    - Por exemplo:

```
myList.insert(0,15)
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Listas: Nesse exemplo a função vai inserir o elemento “20” no final da lista.
  - A lista ficará [15, 1, 2, 20].
  - Esquematicamente, podemos exemplificar da seguinte forma:

```
myList.append(20)
```




```
myList.remove(2)
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Listas: Item de determinada posição:

```
>>> myList
[10, 20, 30, 20, 15, 20, 1, 3]
>>> myList.pop()  # retorna o elemento que está na última posição
3
>>> myList.pop(0) # retorna o elemento que está na  posição 0
10
>>> myList.pop(-1) # retorna o primeiro elemento na ordem inversa
1
>>> myList.pop(20)  # retorna o elemento que está na  posição 20
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    myList.pop(20)
IndexError: pop index out of range
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Listas: Considerando a lista [15, 1, 20], o exemplo anterior estamos verificando se o elemento “1” está na lista. Essa expressão retornará True, pois o elemento está na lista.
  - Se a expressão fosse teria como resultado false, pois o elemento “2” não faz mais parte da lista.

```
2 in myList
```

Estruturas de Dados Básicas

- **Estruturas Lineares**

Exemplo: Supondo que a lista myList seja a lista [15, 1, 20].

O comando

```
myList+myList
```

retornará

```
[15, 1, 20, 15, 1, 20]
```

O comando

```
myList * 3
```

retornará

```
[15, 1, 20, 15, 1, 20, 15, 1, 20]
```

Estruturas de Dados Básicas

- **Estruturas Lineares**

Supondo que a lista myList seja a lista [15, 1, 20, 30, 50].

O comando

```
myList[:2]
```

Retornará a lista da posição inicial até a posição 2, exclusive a posição 2.

```
[15, 1]
```

O comando

```
myList[1:]
```

Retornará a lista da posição 1 até o final.

```
[1, 20, 30, 50]
```

O comando

```
myList[1:2]
```

Retornará a lista da posição 1 até a posição 2, exclusive a posição 2..

```
[1]
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Lista: outras funções

```
for element in myList:
    print(element)
```

```
>>> myList = [1,2,3]
>>> otherList = myList.copy()
>>> otherList
[1, 2, 3]
```

```
>>> myList.clear()
>>> myList
[]
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Lista: outras funções

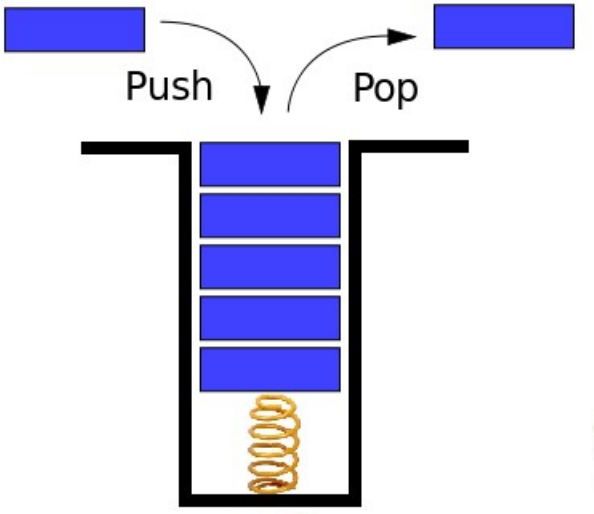
```
>>> myList = [10, 20, 30, 20, 15, 20 ]
>>> myList.count(20)
3

>>> myList.extend([1,3])
>>> myList
[10, 20, 30, 20, 15, 20, 1, 3]

>>> myList
[20, 30, 20, 15, 20]
>>> myList.sort()
>>> myList
[15, 20, 20, 20, 30]
```

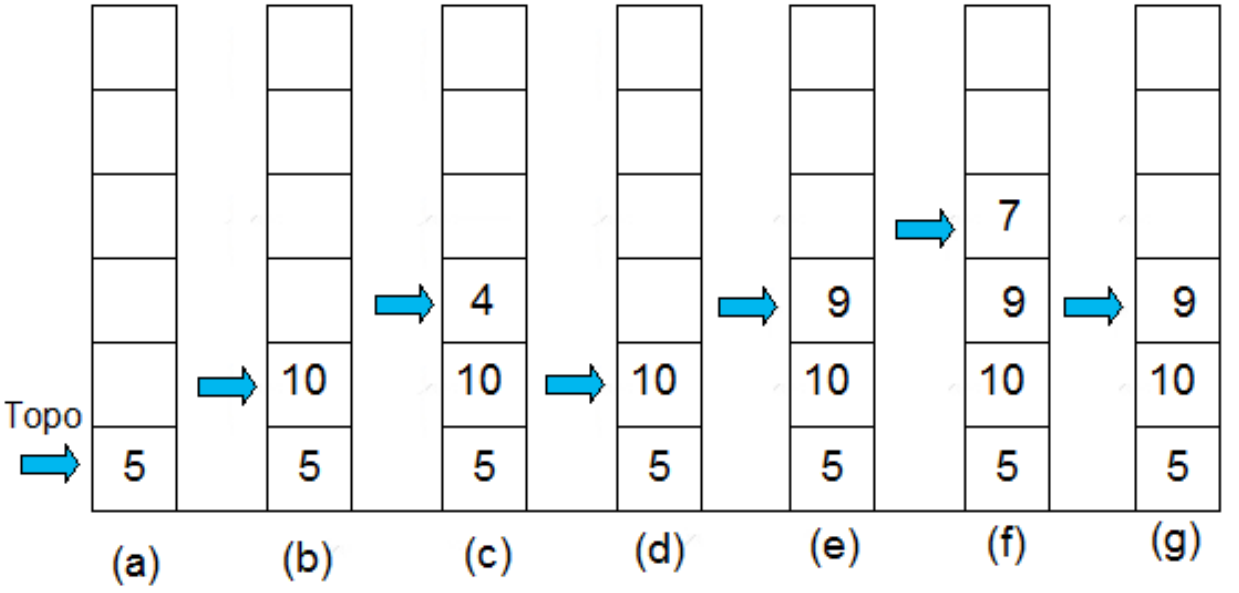
Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:
    - A **pilha** é uma coleção de objetos cujo princípio de inserção e remoção é o ***last-infirst-out*** (LIFO).
    - Este princípio consiste no procedimento de que o último elemento a entrar na estrutura (***last-in***), é o primeiro a sair (***first-out***).



Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:
    - A **pilha** é uma coleção de objetos cujo princípio de inserção e remoção é o ***last-infirst-out*** (LIFO).
    - Este princípio consiste no procedimento de que o último elemento a entrar na estrutura (***last-in***), é o primeiro a sair (***first-out***).



Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:

Tabela 5.1 – Sequência de operações numa Pilha.

OPERAÇÃO	SAÍDA	CONTEÚDO DA PILHA
s.push(5)	-	5
s.push(10)	-	10 5
s.push(4)	-	4 10 5
s.peek()	4	4 10 5
len(s)	3	4 10 5
s.isEmpty	False	4 10 5
s.pop()	4	10 5
push(9)	-	9 10 5
10 in s	True	9 10 5
s.clear()	-	
s.extend([10,5])		5 10

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:
    - A **pilha** é uma coleção de objetos cujo princípio de inserção e remoção é o ***last-infirst-out*** (LIFO).
    - Este princípio consiste no procedimento de que o último elemento a entrar na estrutura (***last-in***), é o primeiro a sair (***first-out***).

(a) (b)

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:

```
pilha = [1, 1, 2, 3, 5]
print("Pilha: ", pilha)

pilha.append(8)
print("Inserindo um elemento: ", pilha)

pilha.append(13)
print("Inserindo outro elemento: ", pilha)

pilha.pop()
print("Removendo um elemento: ", pilha)

pilha.pop()
print("Removendo outro elemento: ", pilha)
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:

```
1  class Stack:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def push(self, item):
9          self.items.append(item)
10
11     def pop(self):
12         return self.items.pop()
13
14     def peek(self):
15         return self.items[len(self.items)-1]
16
17     def size(self):
18         return len(self.items)
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Pilha:

```
20 s=Stack()
21 print(s.isEmpty())
22 s.push(4)
23 s.push(7)
24 s.push(2)
25 print(s.peek())
26 s.push(True)
27 print(s.size())
28 print(s.isEmpty())
29 s.push(8.4)
30 print(s.pop())
31 print(s.pop())
32 print(s.size())
```

Estruturas de Dados Básicas

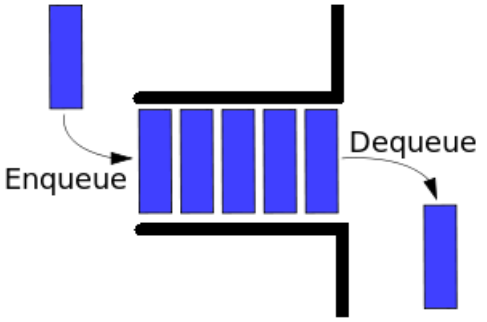
- **Estruturas Lineares**
  - Fila:
    - Coleção de objetos cujo princípio de inserção e remoção é o ***first-in- firstout*** (FIFO).
      - primeiro elemento a entrar na estrutura (***first-in***) é o primeiro a sair (***first-out***).
    - É interessante frisar que o nome da estrutura **fila** veio de uma metáfora da **fila** de pessoas para serem atendidas durante um procedimento qualquer, como por exemplo, a fila de atendimento de um banco.

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila: As operações básicas na estrutura **fila** são inserir um elemento na fila (enqueue), e remover o elemento da fila (dequeue).



Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:
    - A **fila** é uma estrutura muito utilizada em computação como por exemplo:
      - Gerência dos arquivos para impressão
      - Gerência dos processos nas requisições do sistema operacional
      - Buffer para gravações de mídia
      - Processos de comunicação de redes

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:

Tabela 5.1 – Sequência de inserção e remoção numa fila.

OPERAÇÃO	SAÍDA	CONTEÚDO DA FILA
q.enqueue(5)	-	(5)
q.enqueue(10)	-	(5,10)
q.enqueue(4)	-	(5,10,4)
len(q)	3	(5,10,4)
q.dequeue()	5	(10,4)
q.enqueue(9)	-	(10,4,9)
q.enqueue(7)	-	(10,4,9,7)
q.peek()	10	(10,4,9,7)
q.dequeue()	10	(4,9,7)
10 in q	False	(4,9,7)
q.dequeue()	4	(9,7)
q.dequeue()	9	(7)
q.dequeue()	7	()
q.dequeue()	**Error**	()
q.isEmpty	True	-
q.extend([10,20])	-	(10,20)
q.clear()	-	()

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:

Tabela 5.1 – Sequência de inserção e remoção numa fila.

OPERAÇÃO	SAÍDA	CONTEÚDO DA FILA
q.enqueue(5)	-	(5)
q.enqueue(10)	-	(5,10)
q.enqueue(4)	-	(5,10,4)
len(q)	3	(5,10,4)
q.dequeue()	5	(10,4)
q.enqueue(9)	-	(10,4,9)
q.enqueue(7)	-	(10,4,9,7)
q.peek()	10	(10,4,9,7)
q.dequeue()	10	(4,9,7)
10 in q	False	(4,9,7)
q.dequeue()	4	(9,7)
q.dequeue()	9	(7)
q.dequeue()	7	()
q.dequeue()	**Error**	()
q.isEmpty	True	-
q.extend([10,20])	-	(10,20)
q.clear()	-	()

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:

```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def enqueue(self, item):
9          self.items.insert(0,item)
10
11     def dequeue(self):
12         return self.items.pop()
13
14     def size(self):
15         return len(self.items)
16
```

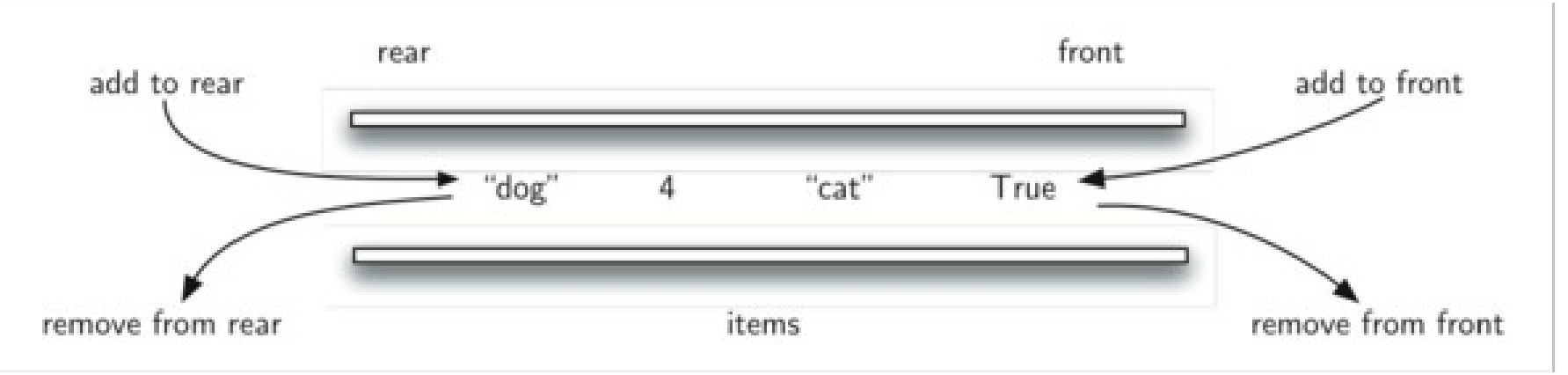
Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila:

```
17 s=Queue()
18 print(s.isEmpty())
19 s.enqueue(4)
20 s.enqueue(7)
21 s.enqueue(2)
22 print("_____")
23 for item in s.items:
24     print(item)
25 print("_____")
26 print(s.size())
27 s.enqueue(22)
28 print(s.size())
29 print(s.isEmpty())
30 print("removi: ", s.dequeue())
31 print(s.size())
```

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila de duas extremidades (deque):
    - Inserção e remoção nas duas extremidades.
    - Capacidades de uma pilha e fila numa mesma estrutura.



Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila de duas extremidades (deque):

Operação	Conteúdo da Deque	Valor Retornado
`d.isEmpty()`	`[]`	`True`
`d.addRear(4)`	`[4]`	
`d.addRear('dog')`	`['dog',4]`	
`d.addFront('cat')`	`['dog',4,'cat']`	
`d.addFront(True)`	`['dog',4,'cat',True]`	
`d.size()`	`['dog',4,'cat',True]`	`4`
`d.isEmpty()`	`['dog',4,'cat',True]`	`False`
`d.addRear(5.4)`	`[5.4,'dog',4,'cat',True]`	
`d.removeRear()`	`['dog',4,'cat',True]`	`5.4`
`d.removeFront()`	`['dog',4,'cat']`	`True`

Estruturas de Dados Básicas

- **Estruturas Lineares**
  - Fila de duas extremidades (deque):

```
class Deque:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.size() == 0

    def addFront(self, item):
        self.items.insert(0,item)

    def addRear(self, item):
        self.items.append(item)

    def removeFront(self):
        return self.items.pop(0)

    def removeRear(self):
        return self.items.pop()

    def rear(self):
        return self.items[self.size()-1]

    def front(self):
        return self.items[0]
```

Atividades

- **Trabalhando com Estruturas Lineares**
  - Atividade:
    - Complete a tabela a seguir considerando o TAD Stack (PILHA) já inicializada com os valores 10,33,5,44. Considere o topo da pilha o elemento 10 e assim sucessivamente.

Onde:

push(e):	Insere o elemento "e", na Pilha.
pop()	Remove o topo da Pilha.
peek()	Retorna o elemento do topo da Pilha.
contains(e):	Retorna **true** se o elemento "e" existe na Pilha e **false** caso contrário..
size()	Retorna o tamanho da Pilha.

OPERAÇÃO	SAÍDA	CONTEÚDO DO PILHA
size()	4	(10,33,5,44)
contains(33)		
push(5)		
pop()		
peek()		
size()		

Atividades

- **Trabalhando com Estruturas Lineares**
  - Atividade:
    - Complete a tabela a seguir considerando o TAD Queue (FILA) já inicializada com os valores 10,33,5,44. Considere o 1º elemento da fila o elemento 10 e assim sucessivamente.

Onde:

offer(e): Insere o elemento "e", na fila.
poll() Remove o 1º elemento da fila.
peek() Retorna o 1º elemento da fila.

OPERAÇÃO	SAÍDA	CONTEÚDO DA FILA
size()	4	(10,33,5,44)
contains(33)		
offer (5)		
poll()		
peek()		
size()