

SME0827 - Estruturas de Dados



Tipos Abstratos de Dados Listas, Filas e Pilhas Aula 09

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Moisés Rocha dos Santos
Monitor:



1



Exemplo de Abstração

- Sistemas computacionais para diferentes usuários, usando dados de automóveis

- Objeto:

- Automóvel

- Usuários:

- Detran: placa, dono atual, débitos, etc.
- Oficina: problema relatado, dono atual, modelo do carro, trabalho realizado, custo, dono atual, etc.
- Revendedora: modelo do carro, ano do modelo, ano de fabricação, preço de venda, preço de compra, etc.

4



Aula de hoje

- Introdução
- Tipos abstratos de dados
- Sequências
- Classes e objetos
- Listas
 - Operações em listas
- Pilhas e filas

© André de Carvalho - ICMC/USP

2

2



Operações sobre dados

- De acordo com o usuário, apenas algumas operações sobre os dados fazem sentido
- Operações que fazem sentido dependem das características consideradas importantes presentes nos dados
 - Calcular lucro obtido com a venda do carro
 - Calcular o valor total das taxas devidas
 - Calcular custo para consertar o carro

André de Carvalho - ICMC/USP

5

5



Abstração

- Para aplicar computação a um problema real de Ciência de Dados, precisamos representar o problema
 - Extrair características importantes dos dados
 - Importância depende do que se pretende fazer com os dados
 - As outras características não precisam ser conhecidas
 - Podem até ser escondidas


3



Abstração

- Processo de descartar os detalhes (características) sem importância de um objeto
 - Manter apenas as características apropriadas para descrever o objeto
- Dados descrevendo o objeto
 - Junto com as operações sobre os dados, formam os tipos abstratos de dados (TADs)

6




Tipos concretos de dados

- Tipos predefinidos de uma linguagem de programação
 - Ex.: *inteiro, caracter, vetor, registro, etc.*
- Tipos concretos de dados (TCDs) são definidos por:
 - Valores que podem assumir e operações que podem ser aplicadas a esses valores
- Definidos pela representação da estrutura de dados

© André de Carvalho - ICMC/USP 7

7




Aspectos positivos de TADs

- Simplicidade:
 - Escondendo a representação interna do usuário, tem menos detalhes para usuário entender
- Flexibilidade
 - Por ser definida pelo seu comportamento, implementador pode mudar a representação
- Segurança
 - Interface age como uma parede, protegendo a implementação do usuário e vice-versa

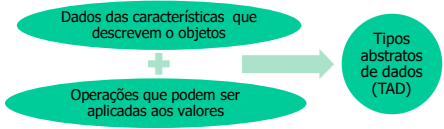
André de Carvalho - ICMC/USP 10

10




Tipos abstratos de dados

- Definem um tipo pelo seu comportamento ao invés de sua representação
 - Geralmente representados por uma interface
 - Exporta o TAD junto com uma coleção de funções (métodos) que definem o seu comportamento



André de Carvalho - ICMC/USP 8

8




TADs em Python

- Python tem vários TADs pre-definidos, vários no formato de sequências
 - Range
 - List (lista)
 - Stack (pilha)
 - Queue (fila)
 - Deque (*Doubleended queue*)
 - Priority queue
 - Tuple (tupla)
 - Dictionary (Dicionário)
 - Set (Conjunto)

© André de Carvalho - ICMC/USP 11

11



Tipos abstratos de dados

- Entidade matemática que define estruturas de dados distinguindo:

Especificação

 - Que valores a estrutura de dados pode assumir e quais as possíveis operações sobre esses valores


Interface

Implementação

 - Como os valores e as operações são implementados

© André de Carvalho - ICMC/USP 9

9




Sequências

- Armazenam coleção ordenada de valores relacionados
- Python possui 2 tipos de sequências especiais
 - Tipo sequência de texto
 - Possui métodos para criar e manipular *strings*
 - Tipo sequência binária
 - Possui métodos para manipular os tipos bytes e bytearray

© André de Carvalho - ICMC/USP 12

12




Sequências

- Principais características
 - Teste de pertinência
 - Operadores *in* (pertence a) e *not in*
 - Operações de indexação
 - Permitem acesso direto a um item de uma sequência
 - Ex.: sequência [posição do item]
 - Operação de fatiamento
 - Permite recuperar uma fatia de uma sequência

© André de Carvalho - ICMC/USP 13

13



Função range


- Por default começa no 0 e aumenta de 1
 - É possível também definir início e incremento da sequência ([ini,] fim[, inc])

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

ini: início da sequência
fim: final da sequência (não inclui valor de fim)
inc: diferença entre dois números consecutivos da sequência

© André de Carvalho - ICMC/USP 16

16




Função range

- Facilita repetição usando uma sequência de números
- Retorna um objeto que parece uma lista, mas não é
 - Retorna os itens que permitem gerar a lista
 - Faz isso para salvar espaço
 - Ex.: >>> *range(10)* ou *range(0, 10)*

© André de Carvalho - ICMC/USP 14

14



Função range


- Por default começa no 0 e aumenta de 1
 - É possível também definir início e incremento da sequência ([ini,] fim[, inc])

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

ini: início da sequência
fim: final da sequência (não inclui valor de fim)
inc: diferença entre dois números consecutivos da sequência

© André de Carvalho - ICMC/USP 17

17




Função range

- Retorna um objeto iterável
 - Pode ser usado por funções e permite obter itens sucessivos
 - É o que faz o comando *for* quando usa a função *range*
 - Ex.: >>> *list(range(10))*
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 - Valor (limite) fornecido nunca faz parte da lista gerada

© André de Carvalho - ICMC/USP 15

15



Função range

- Sequência para iteração não precisa ser numérica
 - Combinar funções *range* e *len*

```
>>> a = ['Maria', 'tinha', 'uma', 'febre', 'alta']
>>> for i in range(len(a)):
...     print(i, a[i])
... 
```

© André de Carvalho - ICMC/USP 18

18



Função range

- Sequência para iteração não precisa ser numérica
- Combinar funções *range* e *len*

```
>>> a = ['Maria', 'tinha', 'uma', 'febre', 'alta']
>>> for i in range(len(a)):
...     print (i, a[i])
...
0 Maria
1 tinha
2 uma
3 febre
4 alta
```

© André de Carvalho - ICMC/USP

19

19



Exemplo

- Seja a lista coisas = [3, 'b', 3, 2.1], determinar o valor de:
 - coisas[0]: 3
 - coisas[3]: 2.1
 - coisas[-2]: 'b'

© André de Carvalho - ICMC/USP

22

22



Listas

- Coleção ordenada de valores (items)
 - Ex.: *minhaLista* = ['Estatistica', 'Programacao']
- Valores podem ser de qualquer tipo e tipos podem estar misturados
 - Ex.: [3, 54, 2], ['casa', 'pe'], [3, 'b', 3, 2.1]
 - Lista sem items é uma lista vazia
 - Ex.: []

© André de Carvalho - ICMC/USP

20

20



Listas

- Tipo de dado mutável (alterável)
 - Pode mudar conteúdo (valor) e tamanho
- Podem implementar vetores e matrizes
- Existem várias semelhanças entre listas e *strings*
 - Ex.: operações de indexação e fatiamento
 - Mas, diferente de listas, *Strings* são imutáveis

© André de Carvalho - ICMC/USP

23

23



Exemplo

- Seja a lista coisas = [3, 'b', 3, 2.1], determinar o valor de:
 - coisas[0]:
 - coisas[3]:
 - coisas[-2]:

© André de Carvalho - ICMC/USP

21

21



Listas

- Estrutura de dados (tipo, classe) list inclui métodos para
 - Consultar (procurar) itens em uma lista
 - Alterar
 - Composição de uma lista
 - Ordem dos itens de uma lista
 - Adicionar itens a uma lista
 - Remover itens de uma lista

© André de Carvalho - ICMC/USP

24

24



Classes e objetos

- Representam um tema por si só
- Mas alguns conceitos básicos são importantes para entender listas
- Objeto é uma instanciação de uma classe
 - Lista é um exemplo de uso de classes e objetos
 - Tipos simples são outros exemplos
 - Ex.: comando $x = 6$
cria um objeto x da classe (tipo) *int*

© André de Carvalho - ICMC/USP

25

25



Classes e objetos

```
>>> minhaLista = ['Estatistica', 'Programacao']
>>> minhaLista
['Estatistica', 'Programacao']
>>> minhaLista.append("Calculo")
['Estatistica', 'Programacao', 'Calculo']
```

© André de Carvalho - ICMC/USP

28

28



Classes e objetos

- Uma classe pode ter métodos (funções) relativos àquela classe apenas
 - Esses métodos podem ser usados somente por objetos daquela classe
 - Ex.: Para a classe list, Python possui o método *append*
 - Adiciona itens ao final de uma lista
 - Ex.: `minhaLista.append("Calculo")`

© André de Carvalho - ICMC/USP

26

26



Classes e objetos

- Uma classe pode também conter campos
 - Variáveis definidas para aquela classe
 - Campos podem ser usados apenas pelos objetos da classe
 - Ex.: `minhaLista.cursoAtual("SME007")`

Funciona como
uma variável local

© André de Carvalho - ICMC/USP

29

29



Classes e objetos

```
>>> minhaLista = ['Estatistica', 'Programacao']
>>> minhaLista
>>> minhaLista.append("Calculo")
```

© André de Carvalho - ICMC/USP

27

27



Relembrando...

- Assim, uma classe é como um tipo
 - Pode conter métodos
 - Que são funções
 - Pode conter campos
 - Equivalem a variáveis
- Um objeto é equivalente a um valor (instanciação) de uma classe

© André de Carvalho - ICMC/USP

30

30

Exemplo

```
# Minha lista de compras
listaCompras = ['abacate', 'manga', 'cenoura', 'banana']
print('Eu tenho', len(listaCompras), 'itens para comprar.')
print('Esses itens sao', end=' ')
for item in listaCompras:
    print(item, end=' ')
print("\n Eu tambem preciso comprar rucula.")
listaCompras.append('rucula')
print('Minha lista de compras agora eh ', listaCompras)
print('You ordenar minha lista por ordem alfabetica')
listaCompras.sort()
print('Minha lista de compras ordenada eh ', listaCompras)
print('O primeiro item que eu vou comprar eh ', listaCompras[0])
itemAntigo = listaCompras[0]
del listaCompras[0]
print('Eu comprei', itemAntigo)
print('Minha lista de compras agora eh ', listaCompras)
```

© André de Carvalho - ICMC/USP

31

Relembrando...

- Supor um objeto do tipo (classe) list, denominado *coisas*
 - Classe list possui vários métodos para:
 - Consultar objetos de uma lista
 - Alterar objetos de uma lista
 - Expandir uma lista
 - Reduzir uma lista

© André de Carvalho - ICMC/USP

34

Exemplo

```
# Minha lista de compras
listaCompras = ['abacate', 'manga', 'cenoura', 'banana']
print('Eu tenho', len(listaCompras), 'itens para comprar.')
print('Esses itens sao', end=' ')
for item in listaCompras:
    print(item, end=' ')
, end = ' '
Ao invés de pular de linha,
no próximo print, inclui
espaço após o string
```

Saida:
Eu tenho 4 itens para comprar.
Esses itens sao abacate manga cenoura banana

```
listaCompras = ['abacate', 'manga', 'cenoura', 'banana']
```

Objeto da classe list

© André de Carvalho - ICMC/USP

32

Métodos para consultar listas

- *coisas.count(x)*
 - Retorna quantas vezes o valor *x* aparece em *coisas*
- *coisas.index(x)*
 - Retorna a posição do valor *x* na lista *coisas*
- *coisas[i]*
 - Retorna item na posição *i* da lista *coisas*
- *len(coisas)*
 - Retorna quantos itens tem a lista *coisas*

© André de Carvalho - ICMC/USP

35

Pausa



© André de Carvalho - ICMC/USP

33

Exemplo

```
>>> coisas = [66.25, 333, -1, 333, 1, 1234.5, 333, 0, 1, 2]
>>> print coisas.count(333), coisas.count(66.25), coisas.count('x')

>>> coisas.index(333)

>>> a = coisas[2]
>>> a

>>> len(coisas)
```

© André de Carvalho - ICMC/USP

36

Métodos para alterar listas

- `coisas.sort()`
 - Ordena os itens da lista `coisas` em ordem crescente
- `coisas.reverse()`
 - Inverte a ordem dos itens da lista `coisas`
- `coisas[i] = x`
 - Armazena valor de `x` na posição `i` da lista `coisas`

© André de Carvalho - ICMC/USP

37

37

Exemplo 1

```
>>> coisas = [66.25, 333, 333, 1, 1234.5]
>>> coisas

>>> coisas.insert(2, -1)
>>> coisas.append(333)
>>> coisas

>>> ltb = [0, 1, 2]
>>> coisas.extend(ltb)
>>> coisas
```

© André de Carvalho - ICMC/USP

40

40

Exemplo

```
>>> coisas

>>> coisas.reverse()
>>> coisas

>>> coisas.sort()
>>> coisas

>>> coisas[2] = 3
>>> coisas
```

© André de Carvalho - ICMC/USP

38

38

Exemplo

```
# Minha lista de compras
listaCompras = ['abacate', 'manga', 'cenoura', 'banana']
print('Eu tenho', len(listaCompras), 'itens para comprar.')
print('Esses itens são', end='')
for item in listaCompras:
    print(item, end=' ')
print('\n Eu também preciso comprar rucula.')
listaCompras.append('rucula')
print('Minha lista de compras agora é ', listaCompras)
print('Você ordenar minha lista por ordem alfabética')
listaCompras.sort()
print('Minha lista de compras ordenada é ', listaCompras)
print('O primeiro item que eu vou comprar é ', listaCompras[0])
itemAntigo = listaCompras[0]
del listaCompras[0]
print('Eu comprei', itemAntigo)
print('Minha lista de compras agora é ', listaCompras)
```

© André de Carvalho - ICMC/USP

41

41

Métodos para expandir listas

- `coisas.append(x)`
 - Anexa item `x` ao final da lista `coisas`
 - `coisas.append(x)` equivale a `coisas[len(coisas):] = [x]`
- `coisas.extend(ltb)`
 - Anexa lista de itens `ltb` ao final da lista `coisas`
 - `coisas.extend(ltb)` equivale a `coisas[len(coisas):] = ltb`
- `coisas.insert(i, x)`
 - Insere item `x` na `i`-ésima posição da lista `coisas`
 - `coisas.insert(len(coisas), x)` equivale a `coisas.append(x)`

© André de Carvalho - ICMC/USP

39

39

Exemplo

```
# Minha lista de compras
...
print('\n Eu também preciso comprar rucula.')
listaCompras.append('rucula')
print('Minha lista de compras agora é ', listaCompras)
print('Você ordenar minha lista por ordem alfabética')
listaCompras.sort()
print('Minha lista de compras ordenada é ',
      listaCompras)
```

Saída:

```
Eu também preciso comprar rucula
Minha lista de compras agora é abacate manga cenoura banana rucula
Você ordenar minha lista por ordem alfabética
Minha lista de compras ordenada é abacate banana cenoura manga rucula
```

```
listaCompras = ['abacate', 'banana', 'cenoura', 'manga', 'rucula']
```

© André de Carvalho - ICMC/USP

42

42



Exercício

- Escrever um módulo em Python que:
 - Recebe como entrada uma lista de números
 - Envia a lista para uma função que soma os valores presentes na lista
 - Retorna soma a quem chamou a função
 - Imprime valor da soma

© André de Carvalho - ICMC/USP

43

43



Exercício

- Dada a lista do exercício anterior, escrever um módulo em Python que:
 - Retira da lista todo objeto cujo valor for menor que 20% da soma dos valores
 - Envia a nova lista para uma função que soma os valores presentes na lista
 - Retorna soma a quem chamou a função
 - Imprime valor da soma

© André de Carvalho - ICMC/USP

46

46



Métodos para reduzir listas

- `coisas.remove(x)`
 - Remove primeiro item da lista `coisas` cujo valor é igual a x
 - Se lista não possui item com valor x , ocorre um erro
- `coisas.pop([i])`
 - Remove e retorna item de `coisas` na posição i
 - Colchete indica que parâmetro é opcional
 - Se posição não é especificada, é removido e retornado último item da lista `coisas`

© André de Carvalho - ICMC/USP

44

44



TAD Pilhas

- A inserção e remoção de elementos seguem o princípio de:
 - Último a entrar é o primeiro a sair (LIFO)
- Métodos do tipo list facilitam implementação de uma pilha
 - Para incluir um item, usar o método `append()`
 - Para recuperar um item, usar o método `pop()`



© André de Carvalho - ICMC/USP

47

47



Exemplo

```
>>> coisas = [66.25, 333, -1, 333, 1, 1234.5, 333, 0, 1, 2]
>>> coisas.remove(333)
>>> coisas

>>> coisas.pop()
>>> coisas
```

© André de Carvalho - ICMC/USP

45

45



Exemplo

```
>>> pilha = [3, 4, 5]
>>> pilha.append(6)
>>> pilha.append(7)
>>> pilha

>>> pilha.pop()
>>> pilha

>>> pilha.pop()
>>> pilha.pop()
>>> pilha
```

5
4
3

© André de Carvalho - ICMC/USP

48

48



TAD Filas

- A inserção e remoção de itens seguem o princípio de:
 - Primeiro a entrar é o primeiro a sair (FIFO)
- Métodos do tipo list facilitam implementação de uma fila
 - Para incluir item, usar o método `append()`
 - Para recuperar item, usar o método `pop()`



© André de Carvalho - ICMC/USP

49

50



Conclusão

- Tipos abstratos de dados
- Sequências
- Listas
- Classes e objetos
- Inclusão de itens em listas
- Remoção de itens em listas
- Consulta de itens de listas
- Pilhas e filas

© André de Carvalho - ICMC/USP

52

52



Exemplo

```
>>> fila = ["Pedro", "Joao", "Jose"]
>>> fila.append("Luiz")      # Luiz chegou
>>> fila.append("Mario")    # Mario chegou
>>> fila.pop(0)

>>> fila.pop(0)

>>> fila
```

© André de Carvalho - ICMC/USP

50

50



Perguntas



© André de Carvalho - ICMC/USP

53

53



Exemplo

```
L = [1,2,3]
for i in range(0,len(L)):
    print (i)  # imprime o indice

L = [1,2,3]
for i in range(0,len(L)):
    print (L[i])  # imprime o item da lista de indice i
```

© André de Carvalho - ICMC/USP

51

51



Exercício

- Suponha que você está montando uma fábrica
 - Precisa comprar itens nas quantidades e preços:
 - 10 máquinas empacotadoras, a 1500,00 cada
 - 100 kits de ferramentas, a 130,00 cada
 - 80 vestimentas a 120,00 cada
 - 40 mesas a 600,00 cada
 - Você tem 30.000,00 para gastar

© André de Carvalho - ICMC/USP

54

54



Exercício

- Implementar um módulo em Python para:
 - Receber os itens, com quantidades e valores e armazenar em uma lista
 - Definir que itens vai comprar com os recursos que você tem
 - Gerar uma nova lista apenas com os recursos que você vai comprar
 - Imprimir a lista, o total gasto e o valor que sobrou
 - Usando pilha e depois usando fila