Listas

SSC0301

Prof. Márcio Delamaro

ICMC USP SÃO CARLOS SSC

O que é?

- Uma lista de compras, uma lista de tarefas a realizar, uma lista de presença.
- Em Python: uma sequência de elementos, que podem ser acessados individualmente.
- Os elementos são acessados pela posição que ocupam dentro da lista.
- O esquema é o mesmo que utilizamos para acessar os caracteres de um string.

Como usar

```
>>> q1 = []
>>> q2 = list()
>>> q3 = [1,2,3]
>>> q4 = ['Cerveja', 'Carne', 'Carvão']
>>> q5 = list(q4)
```

Listas heterogêneas

- Os elementos de uma lista não precisam ter todos o mesmo tipo.

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
```

ICMC USP SÃO CARLOS SSC

Listas heterogêneas

- Os elementos de uma lista não precisam ter todos o mesmo tipo.

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
```

int, float, string, misturados na mesma lista

ICMC USP SÃO CARLOS SSC

Listas heterogêneas

- Os elementos de uma lista não precisam ter todos o mesmo tipo.

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
```

int, float, string, misturados na mesma lista

Elementos podem ser repetidos: 3.14 aparece duas vezes na lista.

Acesso aos elementos

- Podemos acessar os elementos da lista usando um índice, que indica a sua posição.
- Índice inicia em zero

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[0]
2.3
>>> q[3]
'Carvão'
>>> q[-1]
3.14
```

ICMC USP SÃO CARLOS SSC

Acesso aos elementos

- Podemos acessar os elementos da lista usando um índice, que indica a sua posição.
- Índice inicia em zero

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[0]
2.3
>>> q[3]
'Carvão'
>>> q[-1]
3.14
```

Índice negativo (a partir de -1) é contado do final da lista.

Elementos que não existem

- Se tentarmos acessar uma posição que não existe, teremos um erro.

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[6]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

ICMC USP SÃO CARLOS SSC

Adicionar elementos

- Existem funções para manipular elementos da lista
- Adicionar, remover, encontrar etc

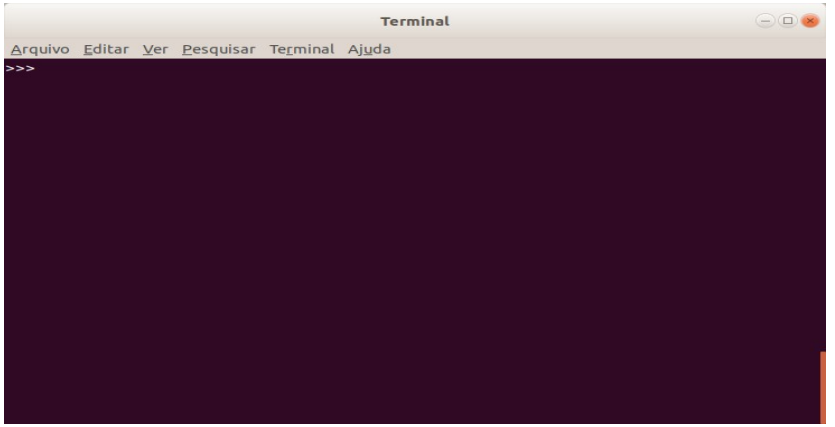
Adicionar elementos

- Existem funções para manipular elementos da lista
- Adicionar, remover, encontrar etc

```
>>> q = []
>>> q.append(3.14)
>>> q.append(18)
>>> q
>>> [3.14, 18]
```

Exemplo

- Ler uma lista de números (float)
- Computar a soma
- Computar a média



ICMC USP SÃO CARLOS SSC

Sublistas

- Podemos criar sublista a partir de listas existentes
- Usamos a mesma notação que usamos para strings
- *[a:b]* – inclui todos os elementos que estão entre o índice *a* (inclusive) e o índice *b* (exclusive)

Sublistas

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[1:4]
[3, 3.14, 'Carvão']
>>> q[-4:4]
[3.14, 'Carvão']
```

Sublistas

- Qualquer um dos dois valores pode ser omitido.

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:]
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:4]
[2.3, 3, 3.14, 'Carvão']
>>> q[-4:]
[3.14, 'Carvão', 7, 3.14]
```

ICMC USP SÃO CARLOS SSC

Sublistas

- Qualquer um dos dois valores pode ser omitido.

`q[a:] = de a até o final`

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:]
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:4]
[2.3, 3, 3.14, 'Carvão']
>>> q[-4:]
[3.14, 'Carvão', 7, 3.14]
```

`q[:b] = do início até b`

Elementos que não existem

- O que acontece se tentarmos usar um índice que não existe na notação de sublista?

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:17]
```

Elementos que não existem

- O que acontece se tentarmos usar um índice que não existe na notação de sublista?

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[:17]
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[-30:17]
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
```

Nesse caso não há erro. Considera-se o final ou o início da lista original para criar a nova lista.

Ainda sublistas

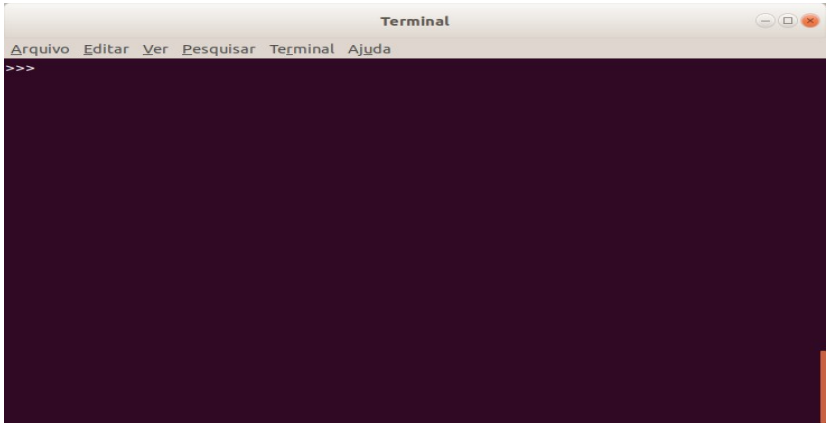
- Terceiro valor indica de quantos em quantos elementos desejamos pegar da lista
- *[1:5:2]* para indicar que desejamos os elementos entre 1 e 5 (excluído), de 2 em 2

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q1 = q[1:5:2]
>>> q1
[3, 'Carvão']
>>> t = list(range(100,120))
>>> t[3:18:3]
[103, 106, 109, 112, 115]
```

ICMC USP SÃO CARLOS SSC

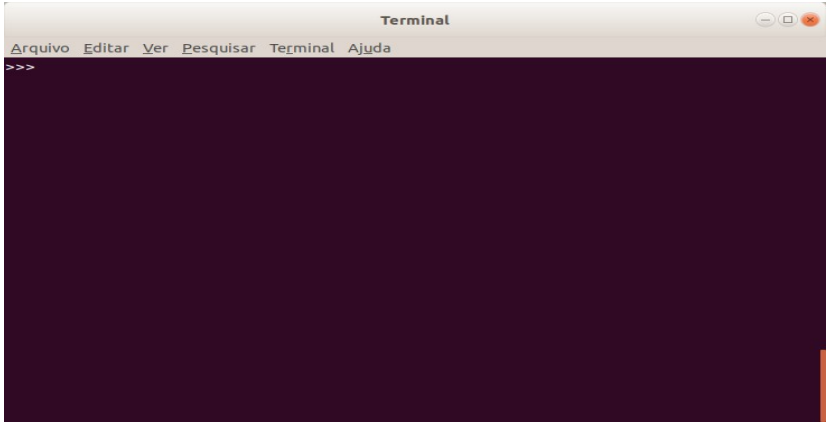
Exemplo

- Ler uma lista de números (float)
- Computar a soma e média dos itens nas posições pares
- Computar a soma e média dos itens nas posições ímpares



Exemplo

- Ler uma lista
- Criar uma segunda lista com os elementos da 1ª lista na ordem invertida



Lista é mutável

- Podemos mudar o valor de uma determinada posição
- Não podemos alterar ou adicionar uma posição que não existe

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q[2] = 'Carne'
>>> q
[2.3, 3, 'Carne', 'Carvão', 7, 3.14]
q[-1] = 0
>>> q
[2.3, 3, 'Carne', 'Carvão', 7, 0]
>>> q[6] = 'Gelo'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Inserir elementos

- Para inserir elementos, devemos usar funções
- q.append(r) – coloca o valor r no final da lista q
- q.insert(k,r) – colocar o valor r na posição k da listar q. Os demais elementos, a partir de k, são deslocados para a “direita” ou para “cima”.

Inserir

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q.append('Gelo')
>>> q
[2.3, 3, 3.14, 'Carvão', 7, 3.14, 'Gelo']
>>> q.insert(2,33)
>>> q
[2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo']
>>> q.insert(10,'Linguiça')
>>> q
[2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo', 'Linguiça']
>>> q.insert(-10, 8)
>>> q
[8, 2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo', 'Linguiça']
```

Inserir

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q.append('Gelo')
>>> q
[2.3, 3, 3.14, 'Carvão', 7, 3.14, 'Gelo']
>>> q.insert(2,33)
>>> q
[2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo']
>>> q.insert(10,'Linguiça')
>>> q
[2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo', 'Linguiça']
>>> q.insert(-10, 8)
>>> q
[8, 2.3, 3, 33, 3.14, 'Carvão', 7, 3.14, 'Gelo', 'Linguiça']
```

Insere no fim

Insere no início

Remover elementos

- q.pop() – retorna o valor do último elemento da lista e tira esse elemento da lista
- q.pop(k) – retorna o valor do elemento k e remove esse elemento da lista
- Se a posição k for inválida, ocorre um erro

Remover

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> x = 2 * q.pop()
>>> q
[2.3, 3, 3.14, 'Carvão', 7]
>>> x
6.28
```

Remover

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> x = 2 * q.pop(2)
>>> q
[2.3, 3, 'Carvão', 7, 3.14]
>>> x
6.28
```

Remover elementos

- q.remove(r) – remove o elemento r que está na lista q
- Remove a primeira ocorrência
- Se não existir r na lista, ocorre um erro

Remover

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q.remove(3.14)
>>> q
[2.3, 3, 'Carvão', 7, 3.14]
>>> q.remove('Gelo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Ainda remover

- Comando `del` usa forma mais intuitiva de lista
- Eliminar valor que está na posição 3 da lista q
  - del q[3]
- Podemos remover uma sublista inteira
  - del q[1:5]

Ainda remover

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> del q[2]
>>> q
[2.3, 3, 'Carvão', 7, 3.14]
>>> del q[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

ICMC USP SÃO CARLOS SSC

Ainda remover

```
>>> t = list(range(100, 120))}
>>> del t[3:18:3]
>>> t
[100, 101, 102, 104, 105, 107, 108, 110, 111, 113,
114, 116, 117, 118, 119]
```

ICMC USP SÃO CARLOS SSC

Outras operações

- len(q) – retorna o número de elementos na lista
- r in q – verifica (True ou False) se existe um elemento r na lista q

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> len(q)
>>> 6
>>> len(q[1:5:2])
2
>>> 'Carvão' in q
True
>>> 'Gelo' in q
False
```

ICMC USP SÃO CARLOS SSC

Outras operações

- len(q) – retorna o número de elementos na lista
- r *in* q – verifica (True ou False) se existe um elemento r na lista q

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> len(q)
>>> 6
>>> len(q[1:5:2])
2
>>> 'Carvão' in q
True
>>> 'Gelo' in q
False
```

```
if ‘Gelo’ in q:
   q.remove(‘Gelo’)
```

Outras operações: comparação

- p == q – duas listas são iguais se elas têm o mesmo tamanho e todos seus elementos são iguais
- p != q – se elas não são iguais
- Comparação < e > – são comparados os 1º.s elementos de cada lista e daí se tem o resultado. Mas se eles forem iguais, comparam-se os 2º.s , até que sejam diferentes ou até que termine uma das listas.

Outras operações: comparação

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q == [2.3, 3, 3.14, 'Carvão', 7, 3.14]
True
>>> q < [3, 3.14, 'Carvão', 7, 3.14]
True
>>> q > [2, 3, 3.14, 'Carvão', 7, 3.14]
True
>>> q < [2.3, 3, 3.14, 'Carvão', 7, 3.14, 'Gelo']
True
```

ICMC USP SÃO CARLOS SSC

Outras operações

- **p + q** : produz uma nova lista que contém os elementos da lista p, seguidos dos elementos da lista q (concatenação)
- **k * p** : k é um inteiro. Produz uma nova lista em que os elementos da lista p são repetidos k vezes

Outras operações

```
>>> q = [2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> q + q
[2.3, 3, 3.14, 'Carvão', 7, 3.14, 2.3, 3, 3.14, 'Carvão', 7,
3.14]
>>> q + ['Gelo']
[2.3, 3, 3.14, 'Carvão', 7, 3.14, 'Gelo']
>>> q[1:5:2] + q
[3, 'Carvão', 2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> [] + q + []
[2.3, 3, 3.14, 'Carvão', 7, 3.14]
>>> 2 * q
[2.3, 3, 3.14, 'Carvão', 7, 3.14, 2.3, 3, 3.14, 'Carvão', 7,
3.14]
```