PYTHON – LISTAS

Introdução à Programação

SI1

Conteúdo

- Listas
  - Conceitos
  - Operações
  - Métodos
  - Exercícios

Por que usar sequências?

- Escreva um programa que pede 4 times ao usuário em ordem de classificação no campeonato. Depois peça ao usuário para informar uma posição e seu programa deve imprimir o time que está naquela posição.

- E se forem 50 times?

- E se for uma quantidade indefinida de times, até que o usuário digite “fim”?

Sequências

- Podem ser indexados por algum valor ordinal posicional
- Algumas operações são aplicadas a todos os tipos de sequências.

- Listas
  - li = [1,2,3, ‘abc’]

- Tuplas
  - li = (23, ‘abc’, 4.56, (2,3), ‘def’)

- Strings
  - st = “Hello World” st = ‘Hello World’

Sequências

- Manipulando sequências!
  - Pelo índice a partir de 0 Ex: ti [0]
  - Índices podem ser positivos ou negativos!
  - Ex: ti[1] (esq.) ti[-4] (dir.)
- Fracionamento e matrizes!
  - li[1:3] , L[1:]
  - matrix = [[1,3,4] , [3,5,6] , [7,8,9]]
- Operador *in*
  - retorna um booleano. Checa se um valor está em uma sequência!
    - 4 in li

Listas

- Listas são coleções **heterogêneas** de objetos, que podem ser de qualquer tipo, inclusive outras listas.
- As listas no Python são **mutáveis**, podendo ser alteradas a qualquer momento
  - é possível fazer atribuições a itens da lista
- Listas podem ser **"fatiadas"** da mesma forma que as ***strings***

Listas

- Uma lista é na verdade um objeto da classe chamada **`list`**

- Na verdade, pode ser vista como uma implementação de ***arrays***
  - Acesso **seqüencial** e **direto** através de índices

Listas

- Listas são **variações de seqüências** assim como strings e portanto têm **APIs semelhantes**
  - Podem ser indexadas e fatiadas
  - Podem ser **concatenadas (+)** e **repetidas**

Operações

- **# Uma nova lista: lista de frutas**

```
lista = ['Caju', 'Laranja', "Banana", 'Uva']
```

- **# Varrendo a lista inteira**

```
for fruta in lista:
    print(fruta)
```

Caju
Laranja
Banana
Uva

Atribuições

```
>>> lista = [2, 28, 9, 'league of legends', 78, 12]
>>> lista[0] = 33
>>> lista
```

[33, 28, 9, 'league of legends', 78, 12]

```
>>> lista[-1] = "teste"
>>> lista
```

[33, 28, 9, "league of legends", 78, "teste"]

```
>>> lista[3] = 99
>>> lista
```

[33, 28, 9, 99, 78, "teste"]

Listas

```
>>> a = [1, 2, 3, 4, 5]    #criação da lista
>>> a[0]
1
>>> a [2]
3
>>> a[-1]
5
>>> a[-3]
3
>>> a[1:]
[2, 3, 4, 5]
>>> a[:3]
[1, 2, 3]
>>> a[1: 4: 2]  #acrescido o passo, coleta-se pulando de 2 em 2
[2, 4]
>>> a[: : -1]
[5, 4, 3, 2, 1] #passo negativo inverte a sequência
```

Operações

- Trocando elementos

```
lista[-1] = 'Laranja'
lista[2]  = 'Uva'
for fruta in lista:
    print(fruta)
```

Caju
Laranja
Uva
Laranja

Operações

- Incluindo elementos

```
lista.append('Melancia')
for fruta in lista:
    print(fruta)
```

Caju
Laranja
Uva
Laranja
Melancia

Operações

- Removendo elementos (por valor)

```
lista.remove('Melancia')
for fruta in lista:
    print(fruta)
```

Caju
Laranja
Uva
Laranja

Operações

- Removendo elementos (por posição)

```
>>> del lista[2]
>>> lista
```

['Caju', 'Laranja', 'Laranja']

Operações

- Ordenando a lista:

```
lista.sort()
for fruta in lista:
    print(fruta)
```

```
Banana
Caju
Laranja
Uva
```

- Invertendo a lista:

```
lista.reverse()
for fruta in lista:
    print(fruta)
```

```
Uva
Laranja
Caju
Banana
```

Operações

- Imprimindo com a posição

```
for i, p in enumerate(lista):
    print(i + 1, '=>', p)
```

1 => Caju
2 => Laranja
3 => Banana
4 => Uva

Observações

- A função ***enumerate()*** retorna dois elementos a cada iteração: a **posição** sequencial e um **item** da seqüência correspondente
- A operações de ordenação (***sort***) e inversão (***reverse***) são realizadas na própria lista, alterando-a

Operações

- “fatiando”

```
>>> print(lista[1:])
['Laranja', 'Banana', 'Uva']
>>> print(lista[:2])
['Caju', 'Laranja']
>>> print(lista[1:3:2])
['Laranja']
>>> print(lista[0:3:2])
['Caju', 'Banana']
```

Operações em Listas

Qual será o valor de b?

```
>>> a = [1,2,3]
>>>  b = a
>>> a.append(4)
>>> print (b)
```

Operações em Listas

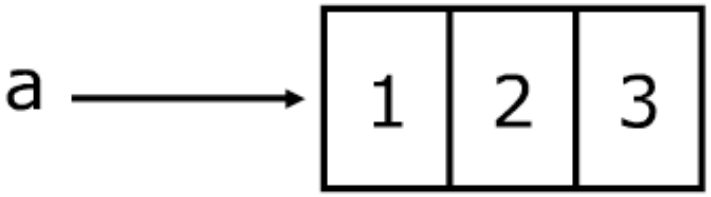
Qual será o valor de b?

```
>>> a = [1,2,3]
>>>  b = a
>>> a.append(4)
>>> print (b)
```

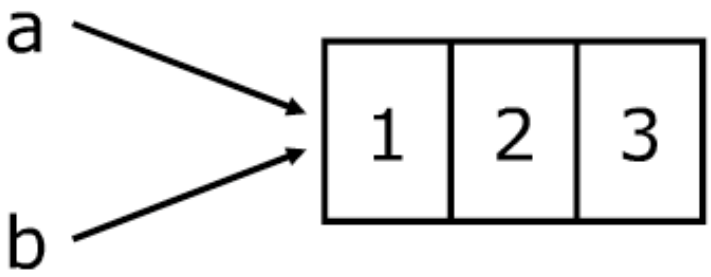
Surpresa!

Operações em Listas

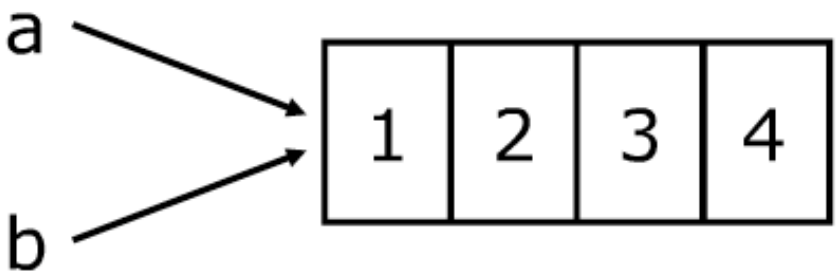
```
a = [1, 2, 3]
```




```
b = a
```




```
a.append(4)
```



Operações em Listas

- Para fazer cópias de listas
  - a = b[:] (2 cópias independentes)
  - a = b (os 2 referenciam o mesmo objeto)
- Qual a diferença entre listas e tuplas ?
  - Listas são mutáveis e Tuplas imutáveis!
  - l = [1,‘abc’,4] t = (1,‘abc’,4,5)
- Atribuição em listas e tuplas
  - *list[0] = ‘3‘ ok!*
  - *t[0] = 3 NOK!!! (Deve-se criar uma nova tupla!)*
  - *t = (3, ‘abc’,4,5)*

Tuplas x Listas

- Listas são mais lentas porém mais poderosas que tuplas
  - Listas podem ser modificadas e tem diversos operadores que podem ser utilizados
  - Tuplas são imutáveis e tem menos funcionalidades!

- Para converter entre listas e tuplas ?
  - li = list(t)

```
>>>a = [ 1,2,3,4,5]
>>>tuple(a)
 (1,2,3,4,5)
>>>list(tuple(a))
[ 1,2,3,4,5]
>>>help(tuple) #ler o help..
```

Mais Operações

```
>>> elemento = [1,2,3,4,5]
>>> sum(elemento)
15
>>> len(elemento)
5
>>> max(elemento)
5
>>> min(elemento)
1
```

Métodos

- **`extend(lista2)`**
  - Acrescenta os elementos de **`lista2`** ao **final da lista**
  - Altera a lista original

```
>>> lista = [1,2]
>>> lista.extend([3,4])
>>> lista
[1,2,3,4]
```

Métodos

- `count(elemento)`
  - Retorna **quantas vezes** o elemento aparece na lista

```
>>> lista2 = [1,2,3,1,8,12,7]
>>> lista2.count(1)
2
```

Métodos

- **`index(elemento)`**
  - Retorna o índice da **primeira ocorrência** de **`elemento`** na lista
  - Um **erro** ocorre se **`elemento`** não consta da lista

```
>>> lista3 = [9,8,33,12]
>>> lista3.index(33)
2
>>> lista3.index(22)

Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
    lista3.index(22)
ValueError: list.index(x): x not in list
```

Métodos

- `insert(índice, elemento)`
  - Insere **`elemento`** na lista na posição indicada por **`índice`**
  - Altera a lista original

```
>>> lista4 = [0,1,2,3]
>>> lista4.insert(1, 'dois')
>>> lista4
[0,'dois',1,2,3]
```

Métodos

- **Atribuições** a fatias podem ser usadas para a mesma finalidade do método insert, entretanto, são **menos legíveis**

```
>>> lista5 = [0,1,2,3]
>>> lista5[1:1] = ['dois']
>>> lista5
[0,'dois',1,2,3]
```

```
>>> lista = [1,2,3,4]
>>> lista[1:3] = [0,9,7]
>>> lista
[1,0,9,7,4]
```

Métodos

- **`pop(índice)`**
  - **Remove** da lista o elemento na posição **índice** e o **retorna**
  - Se índice **não for mencionado**, é assumido o **último**

```
>>> lista6 = [1,2,3,4]
>>> lista6.pop(1)
 2

>>> lista6
[1, 3, 4]

>>> lista6.pop()
 4

>>> lista6
[1, 3]
```

String: método `split()`

- Separa uma string em uma lista de strings menores
- Recebe como parâmetro um caractere separador e um número máximo de pedaços (opcional)
- Retorna uma lista de strings, são os pedaços da string original divididos pelo separador.
- Não altera a string original.

String: método `split()`

```
>>> 'www.eupodiatamatando.com'.split('.')
['www','eupodiatamatando', 'com']

>>> '19:16:23'.split(':')
['19', '16', '23']

>>> hora, minuto, segundos = '19:16:23'.split(':')
>>> hora
'19'
>>> minuto
'16'
>>> segundos
'23'
```

Compreensão de listas

- Funcionalidade muito poderosa da linguagem Python
  - Gera uma lista nova aplicando uma função para cada elemento da lista original.
  - Muito usado por programadores Python! (Economia de código!)
- A sintaxe da compreensão de lista usa-se de palavra-chaves:
  - [expression for name in list]

```
>>>s = [ x**2 for x in range(10) ]
>>> m = [len(x) for x in palavras]
```

Compreensão de listas

- Permite também o uso de filtros (deteminam se uma determina expressão deve ser executada sobre um membro da lista)
  - [expression for name in list if filter]

```
>>> x = [x**2 for x in s if x%2 == 0]
>>> m = [i for i in p if i>5]
```

Compreensão de listas

- Você também pode aninhar compreensão de listas!
    - [expression for name in [expression for name in list]]

Listas: Concatenação e Repetição

- O operador + pode ser usado para concatenação e o operador * para repetição
- >>> lista = [0]*4
- >>>lista
- [0, 0, 0, 0]
- >>> lista = lista + [1]*3
- >>> lista
- [0, 0, 0, 0, 1, 1, 1]

Inicializando listas

- Não é possível atribuir a uma posição inexistente de uma lista
- >>> vetor = []
- >>> vetor [0] = 1
- Traceback (most recent call last):
- IndexError: list assignment index out of range
- Se uma lista vai ser usada como vetor, é conveniente iniciá-la
- >>> vetor = [0]*10
- >>> vetor [0] = 3
- >>> vetor
- [3, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Usando None

- No uso de estruturas de dados, às vezes é importante preencher uma posição com um valor “não válido˜.
- A melhor opção para esse uso é emprega o valor especial **None**
  - Não faz parte de tipo nenhum
  - É melhor que usar 0, [] ou uma string vazia
- Útil para criar uma lista “vazia” mas com um número conhecido de posições. Ex.:
- >>> lista = [None]*5
- >>> lista
- [None, None, None, None, None]

A função list

- Pode ser usada para converter uma string numa lista
- É útil pois uma lista pode ser modificada, mas uma string não.
- Para fazer a transformação inversa, pode-se usar o método join
- Ex:

```
>>> lista = list('alo')
>>> list
['a', 'l', 'o']
>>> lista[1] = 'xx'
>>> lista
['a', 'xx', 'o']
>>> ''.join(lista)
'axxo'
```

Alguns métodos da classe list

- sort(cmp=None, key=None, **reverse = False**)
  - Ordena a lista
  - Os argumentos são opcionais. Por default, a lista é ordenada crescentemente
  - Ex:
    - >>> lista = [9, 8, 7, 1, 4, 2]
    - >>> lista.sort()
    - >>> lista
    - [1, 2, 4, 7, 8, 9]

Alguns métodos da classe list

- sort(cmp=None, key=None, reverse = False)
  - É possível obter a ordem inversa, passando *True* para o argumento *reverse*.
  - Ex:
    - >>> lista = [9, 8, 7, 1, 4, 2]
    - >>> lista.sort(reverse=True)
    - >>> lista
    - [9, 8, 7, 4, 2, 1]
  - OBS: A notação acima permite passar um argumento sem especificar os anteriores, mas poderíamos ter escrito:
    - >>> lista = [9, 8, 7, 1, 4, 2]
    - >>> lista.sort(None, None, True)
    - >>> lista
    - [9, 8, 7, 4, 2, 1]

Alguns métodos da classe list

- sort(**cmp=None**, key=None, reverse = False)
  - O argumento *cmp* especifica uma função de comparação
    - É uma função que o sort chama para definir se um elemento é anterior ou posteriror a outro
    - A função a ser passada tem a forma *comp(elem1, elem2)* e deve retornar um inteiro negativo caso *elem1* seja anterior a *elem2*, positivo caso *elem2* seja anterior a *elem1* e zero se tanto faz
  - Ex.:
    - >>> def compara (elem1, elem2):
    - return elem1%10 – elem2%10
    - >>> compara(100, 22)
    - -2
    - >>> lista = [1000, 22, 303, 104
    - >>> lista.sort(compara)
    - >>> lista
    - [100, 22, 303, 104]

Alguns métodos da classe list

- sort(cmp=None, **key=None**, reverse = False)
  - O argumento *key* especifica uma função aplicada a cada elemento
    - Se for passada uma função *f*, em vez de ordenar os elementos baseado em seus valores *v*, ordenada baseado em *f(v)*
  - Ex:
    - >>> lista = [‘abc’, ‘de’, ‘fghi’]
    - >>> lista.sort(key=len)
    - >>> lista
    - [‘de’, ‘abc’, ‘fghi’]

Matrizes

- Listas podem ser usadas para guardar matrizes
- Por exemplo, podemos criar uma matriz identidade de 3x3 com o código

```
m = []
for i in range(3):
    m.append([0]*3)
    m[i][i] = 1
```

- Obs: Não é uma boa idéia iniciar uma matriz assim:

```
m = [[0]*3]*3
for i in range(3): m[i][i] = 1
print m
```

  - Resultado: [[1, 1, 1], [1, 1, 1], [1, 1, 1]]

EXERCÍCIOS

Exercícios

- Mostre-me as seguinte listas, derivadas de:
  - [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
    - Intervalo de 1 a 9
    - Intervalo de 8 a 13
    - Números pares
    - Números ímpares
    - Todos os múltiplos de 2, 3 e 4
    - Lista reversa
    - Razão entre a soma do intervalo de 10 a 15 pelo intervalo de 3 a 9 em float!

Exercícios

1. Ler uma lista de 5 números inteiros e mostre cada número juntamente com a sua posição na lista.
2. Ler uma lista de 10 números reais e mostre-os na ordem inversa.
3. Ler uma lista com 4 notas, em seguida o programa deve exibir as notas e a média.
4. Ler um vetor com 20 idades e exibir a maior e menor.

Exercícios

5. Inicialize uma lista de 20 números inteiros. Armazene os números pares em uma lista PAR e os números ímpares em uma lista IMPAR. Imprima as listas PAR e IMPAR.

6. Faça um programa que receba a temperatura média de cada mês do ano e armazene-as em uma lista. Em seguida, calcule a média anual das temperaturas e mostre a média calculada juntamente com todas as temperaturas acima da média anual, e em que mês elas ocorreram (mostrar o mês por extenso: 1 – Janeiro, 2 – Fevereiro, . . .).

Exercícios

7. Faça um programa que crie uma matriz aleatoriamente. O tamanho da matriz deve ser informado pelo usuário.

8. Faça um programa que crie uma matriz M (com valores informados do usuário) e mostre a matriz com o dobro dos valores lidos (2*M).

Exercícios

9. Faça um programa que leia um número indeterminado de notas. Após esta entrada de dados, faça o seguinte:
   - Mostre a quantidade de notas que foram lidas.
   - Exiba todas as notas na ordem em que foram informadas.
   - Exiba todas as notas na ordem inversa à que foram informadas, uma abaixo do outra.
   - Calcule e mostre a soma das notas.
   - Calcule e mostre a média das notas.
   - Calcule e mostre a quantidade de notas acima da média calculada.

Exercícios

10. Utilizando listas faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:
    - "Telefonou para a vítima?"
    - "Esteve no local do crime?"
    - "Mora perto da vítima?"
    - “Tinha dívidas com a vítima?"
    - "Já trabalhou com a vítima?“

    O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita“; entre 3 e 4 como "Cúmplice" e; 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".

Exercícios

11. Uma empresa de pesquisas precisa tabular os resultados da seguinte enquete feita a um grande quantidade de organizações: "Qual o melhor Sistema Operacional para uso em servidores?" As possíveis respostas são:
    - 1- Windows XP 2- Unix 3- Linux 4- Netware 5- Mac OS 6- Outro
    - Você deve desenvolver um programa em Python que leia as respostas da enquete e informe ao final o resultado da mesma. O programa deverá ler os valores até ser informado o valor 0 (zero), que encerra a entrada dos dados. Não deverão ser aceitos valores além dos válidos para o programa (0 a 6).
    - Os valores referentes a cada uma das opções devem ser armazenados em uma lista. Após os dados terem sido completamente informados, o programa deverá calcular a percentual de cada uma das respostas e informar o vencedor da enquete.

Exercícios

11. (Continuação) O formato da saída foi dado pela empresa, e é o seguinte:
    - Sistemas Operacionais - Votos %
        - Windows XP 1500 17%
        - Unix 3500 40%
        - Linux 3000 34%
        - Netware 500 5%
        - Mac OS 150 2%
        - Outro 150 2%
    - Total de 8800 votos
    - O Sistema Operacional mais votado foi o Unix, com 3500 votos, correspondendo a 40% dos votos.

Bibliografia

- Livro “Como pensar como um Cientista de Computação usando Python” – Capítulo 8
  - http://pensarpython.incubadora.fapesp.br/portal
- Python Tutorial
  - http://www.python.org/doc/current/tut/tut.html
- Dive into Python
  - http://www.diveintopython.org/
- Python Brasil
  - http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython#head5a7ba2746c5191e7703830e02d0f5328346bcaac