PYTHON – LISTAS

Introdução à Programação

SI2

Sequências

- 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’

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

- Uma lista vazia
  - L = []
- Uma lista com três elementos
  - Z = [15, 8, 9]
- Acesso a um elemento da lista
  - Z[1] — 8

Acesso e modificação a uma lista

```
>>> z = [15,8,9]
>>> z[0]
15
>>> z[1]
8
>>> z[2]
9
>>> z[0] = 7
>>> z[0]
>>> z
[7,8,9]
```

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"]

Calcular a média

```
notas = [6,7,5,8,9]
soma = 0
x = 0
while x<5:
   soma += notas[x]
   x+=1
print("Média: %5.2f" % (soma/x))
```

Calcular a média informando valores

```
notas = [0,0,0,0,0]
soma = 0
x = 0
while x<5:
   notas[x] = float(input("Nota %d" % x))
   soma += notas[x]
   x+=1
x=0
while x<5:
   print("Nota %d: %6.2f" % (x, notas[x]))
   x+=1
print("Média: %5.2f" % (soma/x))
```

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

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

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']

Adição de Listas

```
>>> L = []
>>> L = L+[1]
>>> L
[1]
>>> L+= [2]
[1,2]
>>> L+= [3,4,5]
>>> L
[1,2,3,4,5]
```

Fatiando…

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

```
>>> 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

- Imprimindo com a posição

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

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

- A função ***enumerate()*** retorna dois elementos a cada iteração: a **posição** sequencial e um **item** da seqüência correspondente

Operações em Listas

- Qual será o valor de b?
- >>> a = [1,2,3]
- >>> b = a
- >>> a.append(4)
- >>> print b

Operações em Listas

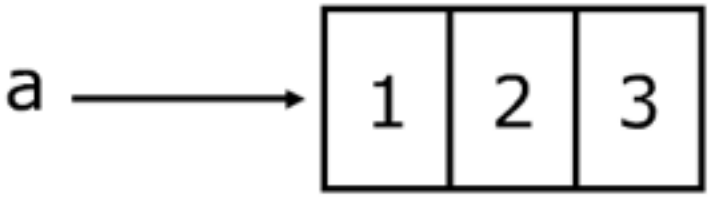
- Qual será o valor de b?
- >>> a = [1,2,3]
- >>> b = a
- >>> a.append(4)
- >>> print b
- [1,2,3,4]

- Surpresa!

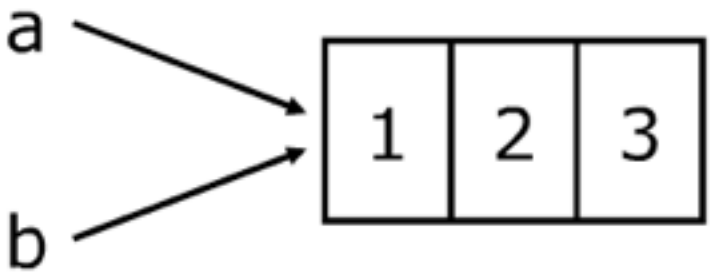
- Dados do tipo listas, dicionarios e pré-definidos pelo usuário são **mutáveis**!

Operações em Listas

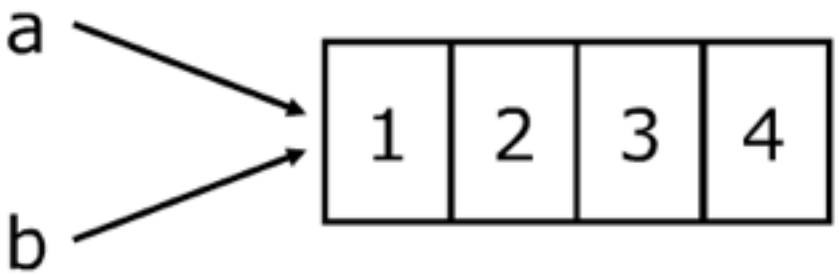
```
a = [1, 2, 3]
```




```
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 é mesmo 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..
```

A função list

- Pode ser usada para converter uma string ou tupla 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'
```

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'
```

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

- **`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]
```

Append vs. extend

```
>>> L = [“a”]
>>> L.append(“b”)
>>> L
[‘a’,’b’]
>>> L.extend([“c”])
>>> L
[‘a’, ‘b’, ‘c’]
>>> L.append([“d”, “e”])
>>> L
[‘a’, ‘b’, ‘c’, [“d”, “e”]]
>>> L.extend([“f”, “g”])
>>> L
[‘a’, ‘b’, ‘c’, [‘d’, ‘e’], ‘f’, ‘g’]
```

Exercícios

1. Faça um programa que leia duas listas e que gere uma terceira com os elementos das duas primeiras

2. Faça um programa que leia duas listas e que gere uma terceira com os elementos das duas primeiras, mas sem repetição

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]
```

Listas como Filas

1. Faça um programa que utilize uma lista para simular uma fila de banco. Inicialmente há 10 clientes na fila, identificados por seu número de atendimento (1-10).
   O programa deve esperar por uma entrada para decidir o que será feito.
   Ao digitar F, um novo cliente é adicionado à fila, com seu respectivo número de atendimento, que deve ser único.
   Ao digitar A, o próximo cliente da fila é atendido.
   Ao digitar S, o programa termina.

Simulação de Fila de Banco

```
ultimo = 10
fila = list(range(1,ultimo+1))
while True:
    print("\nExistem %d clientes na fila" % len(fila))
    print("Fila atual:", fila)
    print("Digite F para adicionar um cliente ao fim da fila,")
    print("ou A para realizar o atendimento. S para sair.")
    operacao = raw_input("operacao (F, A ou S):")
    if operacao == "A":
        if(len(fila))>0:
            atendido = fila.pop(0)
            print("Cliente %d atendido" % atendido)
        else:
            print("Fila vazia! Ninguém para atender.")
    elif operacao == "F":
        ultimo += 1 # Incrementa o ticket do novo cliente
        fila.append(ultimo)
    elif operacao == "S":
        break
    else:
        print("operacao inválida! Digite apenas F, A ou S!")
```

Listas como Filas

1. Altere o programa do slide anterior para que ele aceite vários comandos digitados de uma só vez. Atualmente, apenas um comando pode ser inserido por vez.

   Altere o programa de forma a considerar a entrada de vários comandos como uma string.

   Exemplo: **FFFAAAS** significaria a chegada de três novos clientes, três atendimentos e, finalmente, a saída do programa.

```
ultimo = 10
fila = list(range(1,ultimo+1))
while True:
     print("\nExistem %d clientes na fila" % len(fila))
     print("Fila atual:", fila)
     print("Digite F para adicionar um cliente ao fim da fila,")
     print("ou A para realizar o atendimento. S para sair.")
     operacao = raw_input("operacao (F, A ou S):")
     x=0
     sair = False
     while x < len(operacao):
         if operacao[x] == "A":
             if(len(fila))>0:
                   atendido = fila.pop(0)
                   print("Cliente %d atendido" % atendido)
             else:
                   print("Fila vazia! Ninguém para atender.")
         elif operacao[x] == "F":
             ultimo += 1 # Incrementa o ticket do novo cliente
             fila.append(ultimo)
         elif operacao[x] == "S":
             sair = True
             break
         else:
             print("operacao inválida! Digite apenas F, A ou S!")
         x = x + 1
     if(sair):
        break
```

Listas como Pilhas

1. Faça um programa que utilize uma lista para simular uma pilha de pratos a lavar. Inicialmente há 10 pratos na pilha.
   O programa deve esperar por uma entrada para decidir o que será feito.
   Ao digitar E, um novo prato é adicionado à pilha.
   Ao digitar D, um prato deve ser desempilhado.
   Ao digitar S, o programa termina.

Simulação de Pilha de Pratos

```
prato = 5
pilha = list(range(1,prato+1))
while True:
        print("\nExistem %d pratos na pilha" % len(pilha))
        print("Pilha atual:", pilha)
        print("Digite E para empilhar um novo prato,")
        print("ou D para desempilhar. S para sair.")
        operacao = raw_input("operacao (E, D ou S):")
        if operacao == "D":
                if(len(pilha)) > 0:
                        lavado = pilha.pop(-1)
                        print("Prato %d lavado" % lavado)
                else:
                        print("Pilha vazia! Nada para lavar.")
        elif operacao == "E":
                prato += 1 # Novo prato
                pilha.append(prato)
        elif operacao == "S":
                break
        else:
                print("operacao inválida! Digite apenas E, D ou S!")
```

Listas como Pilhas

1. Faça um programa que leia uma expressão com parênteses. Usando pilhas, verifique se os parênteses estão balanceados, isto é, para cada parêntese aberto há um correspondente fechando. Exemplo:

   (()) **OK**
   ()()(()()) **OK**
   ()) **ERRO**

   Como resolver este problema com pilhas?

```
expressao = raw_input("Digite a sequência de parênteses a validar:")
x=0
pilha = []
while x<len(expressao):
    if(expressao[x] == "("):
        pilha.append("(")
    if(expressao[x] == ")"):
        if(len(pilha)>0):
            topo = pilha.pop(-1)
        else:
            pilha.append(")")  # Força a mensagem de erro
            break
    x=x+1
if(len(pilha)==0):
    print("OK")
else:
    print("Erro")
```

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