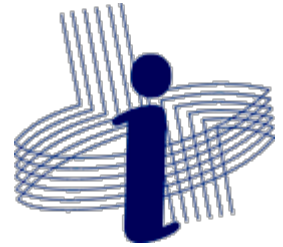


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF101 – Introdução à Programação II

Listas e Tuplas

Listas

- Lista é um tipo composto de dados que permite o armazenamento de vários valores acessados por um índice
- Uma lista pode conter zero (lista vazia) ou mais elementos de mesmo tipo ou de tipos diferentes, podendo, inclusive, conter outras listas
- O tamanho de uma lista é a quantidade de elementos que ela contém



Listas

- Listas são estruturas de dados dinâmicas, isto é, elas podem diminuir ou crescer de tamanho com o tempo
- A lista vazia
 $L = []$
- Uma lista com três elementos
 $L = [34, 5, 21]$
- Neste caso, $L[0] == 34$, $L[1] == 5$, $L[2] == 21$; não existem outros elementos em L



Listas

- Modificação de uma lista
 - Seja L a lista anterior com três elementos, se atribuirmos:
 $L[1] = 85$
 - Então L passa a conter: $[34, 85, 21]$
- **Cuidado:** Não existe $L[3]$, muito menos $L[5]$ ou $L[4]$, etc.!



Cálculo da média

- Seja a lista notas contendo cinco notas de provas, por exemplo:
notas = [5.5, 6.2, 4.3, 7.7, 7.0]
- Então a média das notas pode ser calculada deste modo:

```
soma = 0
```

```
n = 0
```

```
while n < 5:
```

```
    soma = soma + notas[n]
```

```
    n = n + 1
```

```
print("Média: %5.2f" % (soma/n))
```



Cópia de listas

- Cópia independente

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

```
>>> V = L[:]
```

```
>>> V[0] = 6
```

```
>>> L
```

```
[1, 2, 3, 4, 5]
```

```
>>> V
```

```
[6, 2, 3, 4, 5]
```



Cópia de listas

- Cópia compartilhada (*aliasing*)

```
>>> L = [1, 2, 3, 4, 5]
>>> V = L
>>> L
[1, 2, 3, 4, 5]
>>> V
[1, 2, 3, 4, 5]
>>> V[0] = 6
>>> V
[6, 2, 3, 4, 5]
>>> L
[6, 2, 3, 4, 5]
```



Exercício: ordenação de lista

- Um dos problemas clássicos da ciência da computação é a *ordenação* ou *classificação* de um conjunto de dados
- Supondo que os dados estejam estruturados em uma lista homogênea de dados do mesmo tipo e supondo que este tipo suporte ordem, vamos elaborar um método muito simples de ordenação
- Este método é denominado *seleção direta*



Exercício: ordenação de lista

- Supondo uma lista L de tamanho n , a ordenação por seleção direta consiste em:
 1. Selecionar o elemento mínimo $L[j]$ de toda a lista $L[0], L[1], \dots, L[n-1]$
 2. Trocar $L[j]$ com $L[0]$
 3. Repetir os passos 1 e 2 com os conjuntos $L[1], \dots, L[n-1]; L[2], \dots, L[n-1];$ até restar apenas $L[n-1]$
- Implemente em Python, usando funções, o método de ordenação por seleção direta



Fatiamento de listas

- Podemos manipular (acessar) partes (fatias) de uma lista
- O operador que permite o fatiamento é os dois-pontos
- Podemos trabalhar com índices (inteiros) negativos no fatiamento: um índice negativo começa a contar a partir do último elemento
- **Cuidado:** mas o índice negativo do último elemento é -1, do penúltimo é -2, do antepenúltimo é -3 e assim por diante



Fatiamento de listas

- Exemplos

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

```
>>> L[0:5]
```

```
[1, 2, 3, 4, 5]
```

```
>>> L[:5]
```

```
[1, 2, 3, 4, 5]
```

```
>>> L[:-1]
```

```
[1, 2, 3, 4]
```

```
>>> L[1:3]
```

```
[2, 3]
```



Fatiamento de listas

- Exemplos (cont.)

```
>>> L[1:4]
```

```
[2, 3, 4]
```

```
>>> L[3:]
```

```
[4, 5]
```

```
>>> L[:3]
```

```
[1, 2, 3]
```

```
>>> L[-1]
```

```
5
```

```
>>> L[-2]
```

```
4
```



Tamanho de listas

- Obtemos o tamanho corrente de uma lista com a função len

- Exemplos

```
>>> L = [12, 5, 29]
```

```
>>> len(L)
```

```
3
```

```
>>> V = []
```

```
>>> len(V)
```

```
0
```



Cálculo da média (revisão)

- O código que usamos para o cálculo da média de uma lista de números pode ser melhorado:

```
soma = 0
```

```
n = 0
```

```
while n < len(notas):
```

```
    soma = soma + notas[n]
```

```
    n = n + 1
```

```
print("Média: %5.2f" % (soma/n))
```



Inserção de elementos no final

- Uma das vantagens de listas é podermos inserir elementos no final de uma lista
- Para adicionar um elemento no final de uma lista, usamos o método (função) `append`
- Exemplo

```
>>> L = []  
>>> L.append("a")  
>>> L  
['a']
```



Inserção de elementos no final

- Exemplo (cont.)

```
>>> L.append("b")
```

```
>>> L
```

```
['a', 'b']
```

```
>>> L.append("c")
```

```
>>> L
```

```
['a', 'b', 'c']
```



Inserção de elementos no final

- Exemplo

```
L = []
```

```
while True:
```

```
    n = int(input("Digite um número (0 sai): "))
```

```
    if n == 0:
```

```
        break
```

```
    L.append(n)
```

```
i = 0
```

```
while i < len(L):
```

```
    print(L[i])
```

```
    i = i + 1
```



Inserção de elementos em qualquer posição

- Podemos inserir um elemento em qualquer posição em uma lista
- O operador para tal é o método `insert`
- Por exemplo:

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

```
>>> L.insert(2, 8)
```

```
>>> L
```

```
[1, 2, 8, 3, 4, 5]
```



Inserção de elementos em qualquer posição

- A operação `L.insert(i, x)` equivale a um `L.append(x)`, se $i \geq \text{len}(L)$
- Por exemplo:

```
>>> L = [1, 2, 3, 4, 5]
>>> L.insert(10, 8)
>>> L
[1, 2, 3, 4, 5, 8]
```



Concatenação de listas

- Podemos concatenar (juntar) duas listas para formar uma só
- O operador de concatenação é o +
- Exemplo

```
>>> L = [1, 2]
```

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

```
>>> L
```

```
[1, 2, 3, 4, 5]
```



Remoção de elementos

- Podemos remover elementos de uma lista; assim seu tamanho diminuirá
- Para eliminar elementos, usamos o comando `del`
- Exemplo

```
>>> L = [1, 2, 3]
```

```
>>> del L[1]
```

```
>>> L
```

```
[1, 3]
```



Remoção de fatias

- Podemos eliminar fatias inteiras de uma lista de uma vez só com o operador del

- Exemplo

```
>>> L = list(range(101))
```

```
>>> del L[1:99]
```

```
>>> L
```

```
[0, 99, 100]
```



Listas como filas

- Uma *fila* é uma lista que obedece a certas regras de inserção e remoção de elementos
- A inserção é sempre realizada no *fim* da lista e a remoção, no *início* da lista
- Dizemos que o primeiro a chegar é o primeiro a sair: em inglês, esta regra é denominada FIFO (First In First Out)
- No mundo real, o exemplo mais comum de fila é em um banco
- Em Python, o método `pop(i)` retorna o elemento de índice `i` e depois o exclui da lista; observe a diferença com a operação `del`



Simulação de uma 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 = input("Operação (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 = ultimo + 1 # incrementa o tíquete do novo cliente
        fila.append(ultimo)
    elif operacao == "S":
        break
    else:
        print("Operação inválida! Digite apenas F, A ou S.")
```



Listas como Pilhas

- Uma *pilha* é uma lista com uma regra de acesso bem definida: novos elementos são adicionados ao *topo* da pilha; a retirada de elementos também é feita pelo topo
- No mundo real, o exemplo típico é uma pilha de pratos para lavar ou usar
- Em inglês, esta política é chamada de LIFO (Last In First Out)



Simulação de uma 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 = input("Operação (E, D ou S): ")
    if operacao == "D":
        if len(pilha) > 0:
            lavado = pilha.pop(-1) # o topo é o último elemento
            print("Prato %d lavado." % lavado)
        else:
            print("Pilha vazia! Nada para lavar.")
    elif operacao == "E":
        prato = prato + 1 # novo prato
        pilha.append(prato)
    elif operacao == "S":
        break
    else:
        print("Operação inválida! Digite apenas E, D ou S!")
```



Exercício

- Faça um programa que leia uma expressão com parênteses. Usando pilha, verifique se os parênteses foram abertos e fechados na ordem correta
- Por exemplo:

(()) OK

()()(()) OK

()) Errado

((Errado



Listas por compreensão

- Até agora temos criado (definido) listas por *enumeração* de seus elementos
- Python permite outra maneira de criar listas: por *compreensão* de seus elementos
- Esta notação é inspirada na teoria de conjuntos da matemática: definição de conjuntos por compreensão
- Vamos ver que esta notação é muito conveniente também em algumas situações em programação envolvendo listas
- Em Python, para simplificar, denominamos a notação apenas por *listcomp*



Listas por compreensão

- Vamos começar com um exemplo:

```
>>> quadrados = []
```

```
>>> for x in range(10):
```

```
...     quadrados.append(x**2)
```

```
...
```

```
>>> quadrados
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



Listas por compreensão

- Em Python, o exemplo anterior fica mais bem escrito assim:

```
>>> quadrados = [x**2 for x in range(10)]  
>>> quadrados  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```
- Uma listcomp consiste em um par de colchetes contendo uma expressão seguida de uma cláusula for e depois podem vir zero ou mais cláusulas for ou if



Listas por compreensão

- Outros exemplos

```
>>> quadsimps = [x**2 for x in range(10) if x % 2 != 0]
```

```
>>> quadsimps
```

```
[1, 9, 25, 49, 81]
```

```
>>> [x + y for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
```

```
[4, 5, 5, 3, 6, 4, 7]
```

```
>>> from math import pi
```

```
>>> [str(round(pi, i)) for i in range(1, 7)]
```

```
['3.1', '3.14', '3.142', '3.1416', '3.14159', '3.141593']
```



Listas por compreensão aninhadas

- As listcomps podem ser aninhadas, como p.ex., para transpor matrizes de uma maneira concisa:

```
M = [[1, 2, 3, 4],  
      [5, 6, 7, 8],  
      [9, 10, 11, 12]]
```

```
Mtransp = [[linha[i] for linha in M]  
            for i in range(4)]
```



Técnicas de construção de laços

- A função range
 - A função range não retorna uma lista em si, mas, sim, um gerador
- Exemplo 1 (um parâmetro)

```
for v in range(10):  
    print(v)
```
- Exemplo 2 (dois parâmetros)

```
for v in range(5, 10):  
    print(v)
```



Técnicas de construção de laços

- Exemplo 3 (três parâmetros)

```
>>> for v in range(1, 10, 3):
```

```
...     print(v)
```

```
...
```

```
1
```

```
4
```

```
7
```



Técnicas de construção de laços

- Exemplo 4 (três parâmetros)

```
>>> for v in range(7, 0, -3):
```

```
...     print(v)
```

```
...
```

```
7
```

```
4
```

```
1
```



Técnicas de construção de laços

- Transformação do resultado de range em uma lista

```
>>> L = list(range(136, 486, 50))
```

```
>>> print(L)
```

```
[136, 186, 236, 286, 336, 386, 436]
```



Técnicas de construção de laços

- A função `enumerate`
 - Ela gera um par ordenado (2-tupla) em que o primeiro valor é o índice e o segundo é o elemento em si da lista que está sendo enumerada
- Exemplo

```
L = [5, 9, 17]
for x, e in enumerate(L):
    print("%d -> %d" % (x, e))
```



Técnicas de construção de laços

- A função `reversed`
 - Ela faz o gerador ficar em sentido reverso

- Exemplo

```
>>> for i in reversed(range(1, 10, 2)):  
...     print(i)  
...  
9  
7  
5  
3  
1
```



Tuplas

- Tupla é uma estrutura de dados que semelhantemente a lista e *string* constitui uma sequência cujos elementos são acessíveis por meio de índices
- Os elementos de uma tupla podem ser de tipos diferentes
- Depois que uma tupla é criada, seus elementos não podem ser modificados, isto é, uma tupla, assim como *string*, é imutável



Tuplas

- Um valor do tipo tupla é denotado por um par de parênteses e os elementos são separados por vírgula
- Quando não houver ambiguidade, somente as vírgulas são suficientes para denotar uma tupla
- Existe a tupla vazia, denotada apenas por um par de parênteses, sem nada dentro: ()



Tuplas

- Exemplos

```
>>> t = 568, 2.0832, "gato"
```

```
>>> t[0]
```

```
568
```

```
>>> u = t, (1, 2, 3, 4, 5)
```

```
>>> u
```

```
((568, 2.0832, "gato"), (1, 2, 3, 4, 5))
```

```
>>> t[1] = 1.74
```

```
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>TypeError:
'tuple' object does not support item
assignment
```



Tuplas

- Exemplos (cont.)

```
>>> # tuplas podem conter objetos mutáveis
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
>>> v[1][2] = 4
>>> v
([1, 2, 3], [3, 2, 4])
>>> v[0].append(4)
>>> v
([1, 2, 3, 4], [3, 2, 4])
```



Tuplas

- Tupla com apenas um elemento – notação

```
>>> singleton = 365, # observe a vírgula
>>> len(singleton)
1
>>> singleton
(365,)
```
- Tupla vazia – notação

```
>>> vazio = ()
>>> len(vazio)
0
```



Operações com tuplas

- Além do acesso a um elemento, do comprimento de uma tupla e de concatenação, existem as operações de empacotamento (já vista aqui) e de desempacotamento (já usada em INF100)
- Exemplos
 - Empacotamento

```
>>> t = 12345, 54321, "gato"
```
 - Desempacotamento

```
>>> x, y, z = t
```



Operações com tuplas

- Fatiamento de tuplas

```
>>> t = ('a', 'b', 'c', 'd')
```

```
>>> t[1:3]
```

```
('b', 'c')
```

```
>>> t[:-1]
```

```
('a', 'b', 'c')
```

- Repetição dos elementos (concatenação repetida)

```
>>> t * 2
```

```
('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')
```



Operações com tuplas

- Concatenação de tuplas

```
>>> t1 = (1, 2, 3)
```

```
>>> t2 = (4, 5, 6)
```

```
>>> t1 + t2
```

```
(1, 2, 3, 4, 5, 6)
```

- Tuplas podem ser criadas a partir de listas

```
>>> L = [1, 2, 3]
```

```
>>> t = tuple(L)
```

```
>>> t
```

```
(1, 2, 3)
```



Produtos cartesianos

- Podemos criar produtos cartesianos facilmente com listas e tuplas

```
>>> cores = ['preto', 'branco']
>>> tamanhos = ['P', 'M', 'G']
>>> camisetas = [(cor, tam) for cor in
                  cores for tam in tamanhos]

>>> camisetas
[('preto', 'P'), ('preto', 'M'), ('preto',
'G'), ('branco', 'P'), ('branco', 'M'),
('branco', 'G')]
```



Tuplas como registros

- Tuplas podem armazenar registros: cada elemento de acordo com sua posição será um *campo*

```
>>> matr, nome, cargo, sal_por_hora =  
(86275, 'Ana Lima', 'recepcionista',  
45.72)
```
- Dados sobre um(a) funcionário(a) de uma empresa: número de matrícula, nome do(a) funcionário(a), cargo e salário por hora
- Mais tarde, quando virmos a biblioteca `collections`, poderemos implementar *tuplas nomeadas*

