Disciplina	Descrição deste documento
Extra Curricular Algoritmos com implementação em Python 3	Parte 5 Tipos Compostos em Python 3 - Listas

De que se trata?

Ao contrário das variáveis de tipos simples, já vistas, os Tipos Compostos são aqueles que permitem que seu conteúdo seja acessado em conjunto ou em partes. Assim, pode-se dizer que os Tipos Compostos podem ser subdivididos

Motivação

Durante o desenvolvimento de software, independentemente de plataforma e linguagem, é comum a necessidade de lidar com listas. Por exemplo, elas podem ser empregadas para armazenar conjuntos de números inteiros ou reais, palavras, nomes, ou quaisquer outras informações que se queira.

Uma lista é uma estrutura de dados composta por itens organizados de forma linear, na qual cada um pode ser acessado a partir de um índice, que representa sua posição na coleção (iniciando em zero).

Nesta aula o objetivo é começar a trabalhar com listas em Python, linguagem na qual essa estrutura de dados pode armazenar, separadamente e/ou simultaneamente, objetos de diferentes tipos, como strings, números e até mesmo outras listas. Quem conhece outras linguagens pode ficar inicialmente surpreso com a possibilidade que se tem em de misturar informações de variados tipos em uma única lista, mas em Python é assim mesmo.

Para criar uma lista

Pode-se criar uma lista vazia, atribuindo-se [] (abre e fecha colchetes sem nada dentro) a uma variável.

```
L = []
```

Também é possível criar uma lista com alguns itens separados por vírgula.

```
Letras = ['a', 'b', 'c', 'A', 'B', 'C']
Numeros = [14, 5, -6, 27, 12, 34, 25, 18]
Misturado = [23, 5.93, 'texto']
```

Ou também pode-se utilizar a função: list().

```
L = list()                  # equivalente a L = []
L = list(['a', 'b, 'c'])    # equivalente a L = ['a', 'b, 'c']
```

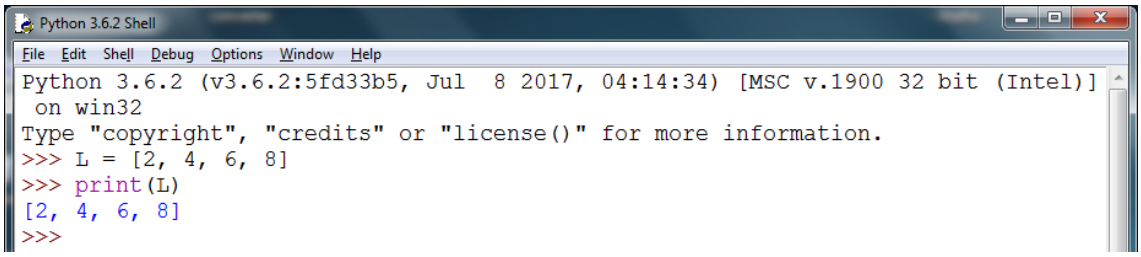
Pode-se criar uma lista através do "list comprehension" que será explicado com mais detalhes em aula.

```
Variavel = [x for x in iterable]

V = [a for a in range(5)]    # gera a lista [0, 1, 2, 3, 4]
M = [x*2 for x in range(3)]  # gera a lista [0, 2, 4]
```

Para acessar individualmente um elemento

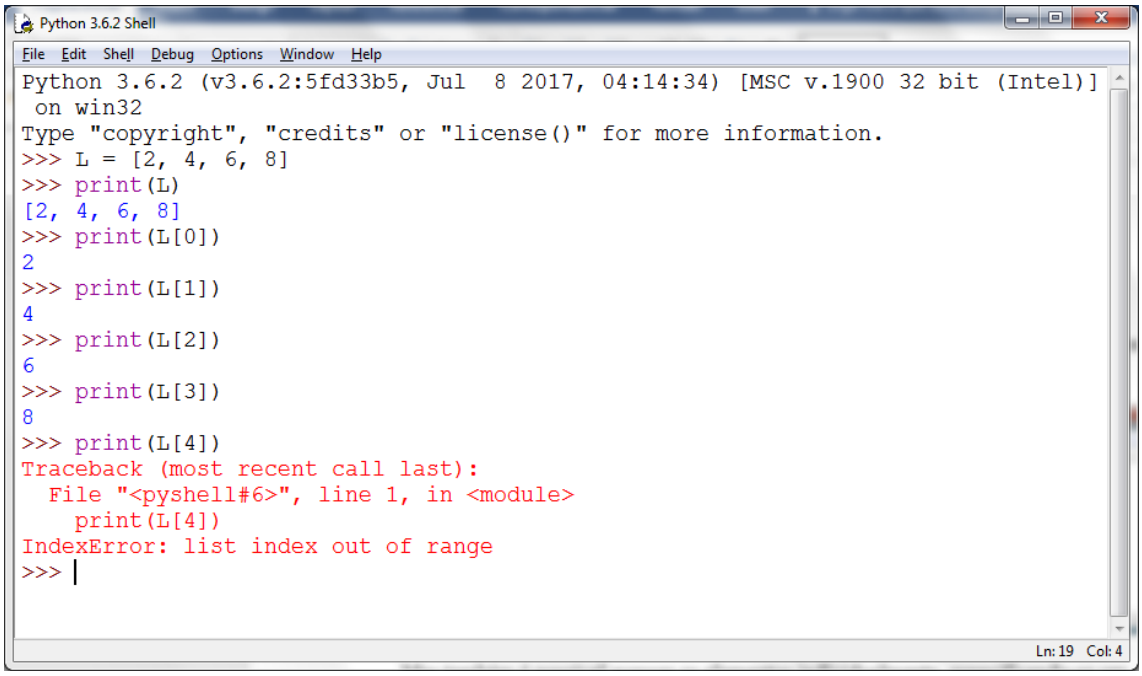
Uma vez criada, a lista permite o acesso a todo o conjunto de dados, por exemplo, pode-se fazer usar o comando print para exibir a lista inteira na tela.



Mas também é possível acessar os elementos individualmente, especificando-se um índice entre colchetes. No exemplo abaixo, dada a lista `L = [2, 4, 6, 8]`, cada um de seus elementos foi acessado individualmente utilizando-se os índices [0], [1], [2] e [3].

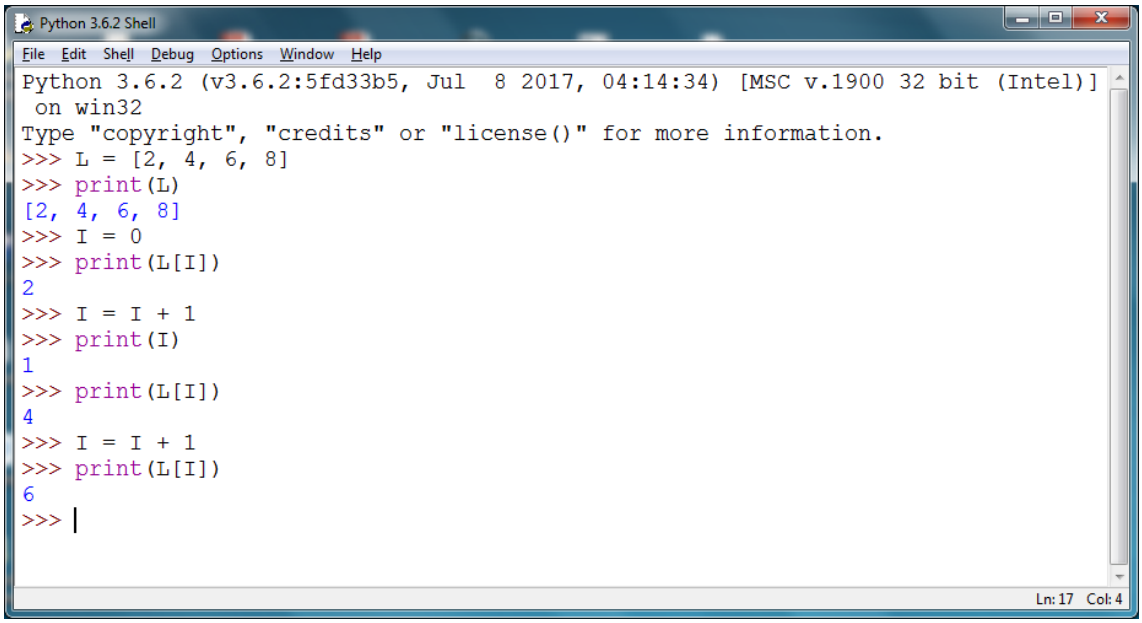
O índice do primeiro elemento da lista é sempre zero [0].

Observe que ao fazer referência a L[4] ocorreu um erro. Isso porque tal elemento não existe. Como a numeração de índice inicia em 0, se a lista tem 4 elementos, então o índice do último é 3.



No exemplo acima foram utilizados índices literais numéricos, ou seja, 0, 1, etc.

Além disso, também é possível usar uma variável do tipo inteiro como índice. Basta definir uma variável que contenha um valor inteiro e substituir os literais numéricos por essa variável. O exemplo a seguir mostra isso.



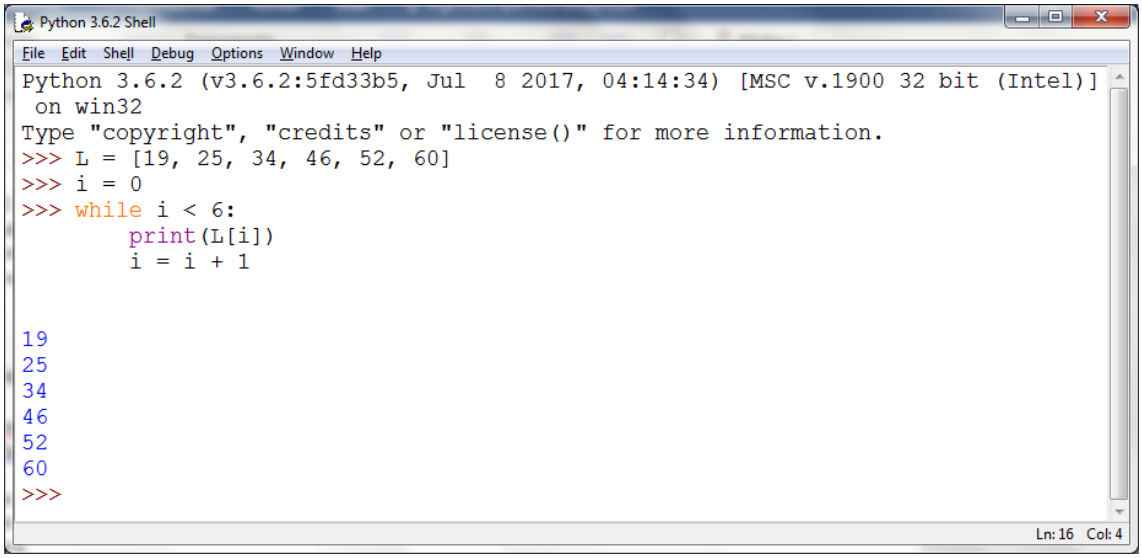
Neste exemplo a lista é a mesma que no anterior. Foi criada a variável I contendo o valor 0 e essa variável foi usada entre colchetes como índice da lista. Desse modo, com I = 0, ao executar o comando print(L[I]) foi exibido na tela o 2, que é o primeiro elemento da lista.

Em seguida somou-se 1 ao I e o print(L[I]) exibiu o valor 4.

Assim, note que ao alterar o valor do índice I é possível ter acesso a um elemento diferente da mesma lista. Isso confere grande flexibilidade ao programador que manipula listas em seus programas.

Percorrendo a lista através de um laço

É muito comum o uso de laços para percorrer a lista, acessando-se todos, ou parte de seus elementos, dentro do laço. Isso é feito no exemplo abaixo, no qual a variável i é usada como índice e, ao mesmo tempo, como variável de controle do laço. Cada elemento da lista é exibido na tela.



Uma outra forma é usar o comando for, que tem uma aplicação diferente no caso de iterações com listas ou tuplas. O comando for permite que uma variável de controle assuma o próprio valor do elemento da lista, um por vez. Abaixo tem-se um exemplo de como percorrer uma lista de forma muito simples.

```
MinhaLista = [2, 4, 6, 8]
for N in MinhaLista:
    print(N)
```

o resultado na tela será:

```
2
4
6
8
```

Neste caso o laço foi executado uma vez para cada um dos elementos da lista, e a cada vez a variável N assumiu o valor de um elemento da lista, seguindo a ordem sequencial existente na lista.

Uso do enumerate(). Abaixo incrementamos a implementação com a função interna enumerate(). Essa função gera uma numeração para os elementos da lista.

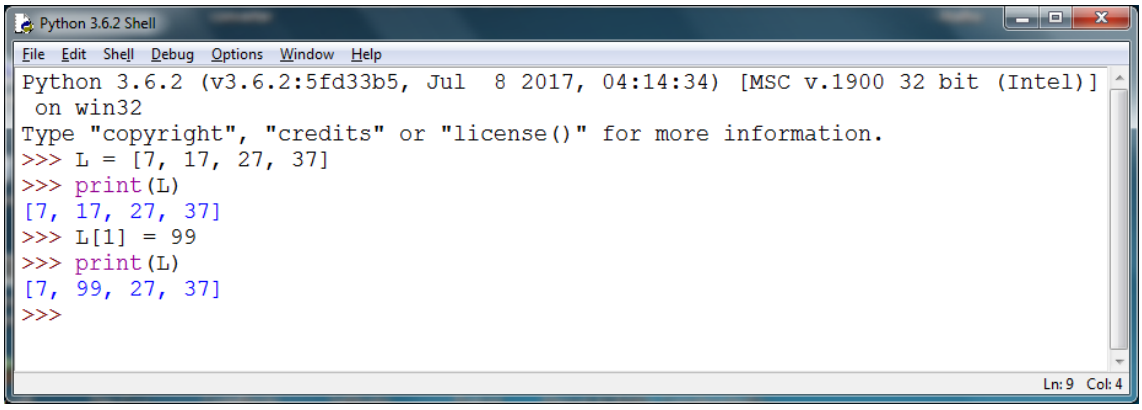
```
Cardapio = ['pizza', 'massa', 'salada', 'churrasco']
print('As possibilidades são:')
for indice, item in enumerate(Cardapio):
    print(indice, item)
```

O resultado na tela será:

```
As possibilidades são:
0 pizza
1 massa
2 salada
3 churrasco
```

Alterando elementos da lista

Uma lista em Python é um objeto mutável. Isso quer dizer que a qualquer momento pode-se alterar um elemento da lista. Para isso basta atribuir um novo valor ao elemento que será acessado por meio do seu índice. Veja o exemplo. Nele, o segundo elemento da lista (cujo índice é 1) teve seu valor 17 trocado por 99.



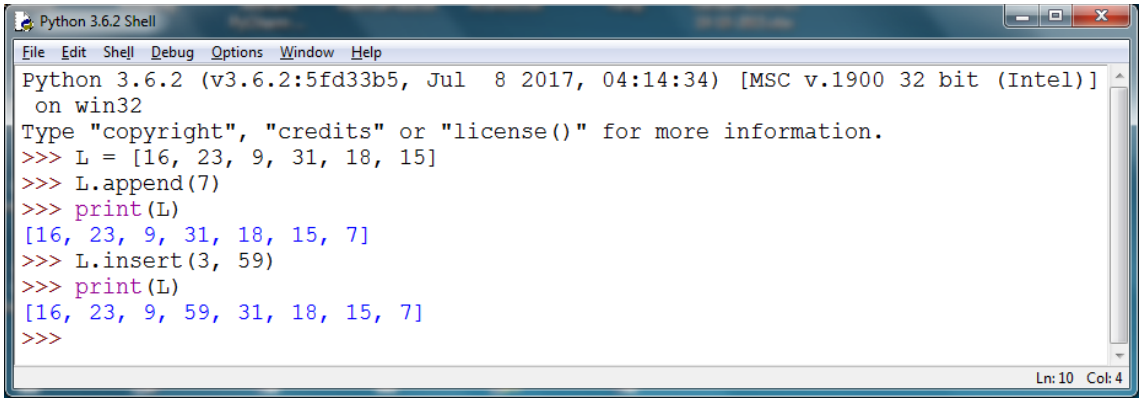
Acrescentando elementos em uma lista

Há duas formas de acrescentar um elemento em uma lista. Para acrescentar elementos ao final da mesma utiliza-se o método append. Para inserção em qualquer ponto da lista usa-se o método insert. Esses métodos são funções nativas agregadas aos objetos da classe list.

Assim, seja a lista `L = [16, 23, 9, 31, 18, 15]`

```
L.append(7)     # acrescenta o valor 7 no final da lista
```

```
L.insert(3, 59)  # acrescenta o valor 59 na posição 3 da lista
```



Removendo elementos em uma lista

Existem três formas de remover um elemento de uma lista. A função del, que é uma função da linguagem Python e se aplica também a outros tipos compostos como os dicionários, por exemplo. E os métodos remove e pop. Procure logo abaixo a explicação de cada um deles.

Copiando listas

Quando se quer copiar uma lista é possível realizar uma simples atribuição, mas cuidado! Haverá uma relação entre as duas, ou melhor, ambas apontam para o mesmo objeto na memória e as alterações em uma afetarão a outra.

```
L1 = [6, 7, 8, 9]
L2 = L1
```

A situação que se tem aqui é que há dois identificadores, L1 e L2, diferentes. Porém ambos referem-se ao MESMO conjunto de dados de modo que se for feita alguma alteração em um elemento de L1, a alteração refletirá em L2 e vice-versa.

Clonando listas

Se quisermos copiar uma lista sem manter a referência entre elas, podemos utilizar o operador de fatiamento [:] (que será devidamente explicado em aula futura).

```
L1 = [6, 7, 8, 9]
L2 = L1[:]
```

Agora L1 e L2 são listas independentes, ou seja, um clone.

Juntando listas

```
m = [1, 2, 3]
n = [4, 5, 6]
o = m + n
print(o) # [1, 2, 3, 4, 5, 6]

o += [7, 8, 9]
print(o) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Funções nativas para listas

append()

```
nums = ["um"]
print(nums) # ['um']
```

```
nums.append("dois")
nums.append("tres")
nums.append("quatro")
print(nums) # ['um', 'dois', 'tres', 'quatro']
```

index()

A função index() retorna o index de determinado elemento.

```
animals = ["ant", "bat", "cat"]
print(animals.index("bat")) # 1
```

Se você procurar por um item que não existe um erro será lançado.

```
print(animals.index("dog")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'dog' is not in list
```

Portanto, se o seu objetivo é saber se um item pertence a lista utilize o operador de teste de inclusão in.

```
>>> animals = ["ant", "bat", "cat"]
>>> 'dog' in animals
False
>>> 'cat' in animals
True
```

insert()

```
animals = ["ant", "bat", "cat"]
animals.insert(1, "dog")
print(animals) # ["ant", "dog", "bat", "cat"]
```

remove()

Remove através do valor, ou seja, se você sabe que quer remover o item "`ant`", mas não sabe em que posição ele está, então deve usar o remove. Caso exista mais de uma ocorrência, remove apenas s primeira.

```
animals = ["ant", "bat", "cat"]
animals.remove("ant")
print(animals) # ["bat", "cat"]
```

pop()

Remove através do índice e retorna o valor removido. O conceito que define o funcionamento do método pop está relacionado com um assunto que será aprendido na disciplina de Estrutura de Dados, no tocante a gerenciamento das estruturas conhecidas como filas e pilhas.

```
animals = ["ant", "bat", "cat"]
animals.pop(0) # 'ant'
print(animals) # ["bat", "cat"]
```

Semelhante é a utilização da função del(), a qual remove pelo índice, porém sem retornar o valor que ocupava a posição:

```
animals = ["ant", "bat", "cat"]
```

```
del(animals[0])
print(animals) # ["bat", "cat"]
```

sort()

```
lista = ["c", "b", "a"]
print(lista) # ['c', 'b', 'a']

lista.sort()
print(lista) # ['a', 'b', 'c']
```

Para ordenar uma lista também é possível utilizar a função interna sorted(), exemplo:

```
sorted([5, 2, 3, 1, 4]) # [1, 2, 3, 4, 5]
```

Uma observação importante é que a função sort() não retorna a lista, então...

```
lista = ["c", "b", "a"]
print(lista.sort()) # None
```

... já a função sorted() retorna a lista ['a', 'b', 'c'] veja:

```
lista = ["c", "b", "a"]
print(sorted(lista))
# ['a', 'b', 'c']
```

Em aula futura veremos como implementar um algoritmo de ordenação. Em Python esse algoritmo, e muitos outros, encontram-se disponíveis e prontos para uso. Porém um estudante de programação não deve se satisfazer com isso. É preciso saber implementar os algoritmos mais comuns, como ordenação, pesquisa sequencial, pesquisa binária, geração de números pseudoaleatórios, entre outros.

Exercícios propostos

1. Escreva um programa que permaneça em laço até que seja digitado o valor zero ou negativo. Cada valor positivo lido deve ser inserido no final de uma lista, usando o método append. Ao final exiba a lista completa na tela.
2. Refaça o programa anterior, porém ao final exiba na tela cada elemento da lista em uma linha da tela (este programa deve exibir um elemento por vez dentro de um laço e usando um índice para acessar cada elemento individualmente).
3. Escreva um programa que leia do teclado uma lista com tamanho de 10 elementos e exiba-a na tela na ordem inversa à ordem de leitura.
4. Escreva um programa que leia do teclado duas listas com tamanho 10, com números inteiros. Em seguida o programa deve juntar as duas listas em uma única com o tamanho 20.