Aula 6 – Listas, Tuplas e Dicionários

Professor Rodrigo Maciel

Variáveis de estrutura de dados (Coleções)

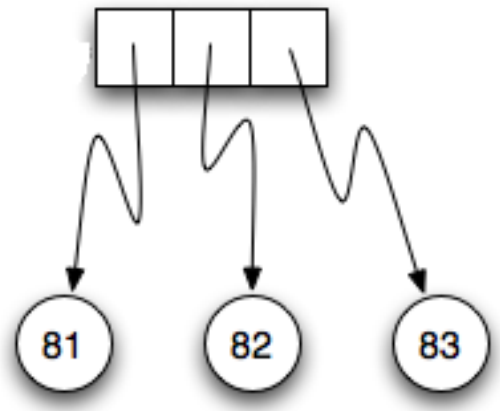
Em python é possível compor estruturas de dados primitivas, e armazená-las em outras estruturas, chamadas de estruturas compostas.

- Listas
- Tuplas
- Dicionários

Listas em python

Listas são cadeias de valores, isto é, armazenam mais de um valor. São estruturas ordenadas, mutáveis e heterogêneas.

- Ordenadas: cada valor tem seu índice, na ordem que estão armazenados.
- Mutáveis: é possível alterar os valores.
- Heterogêneas : valores podem ser de tipos diferentes.



Inicialização de uma lista

- Inicialização de uma lista com valores

Nome_Lista = [valor1, valor2, ..., valorN]

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

- Inicialização de uma lista vazia

```
>>> x = list()
>>> x = []
```

Acessando elementos de uma lista

- O acesso dos elementos são feito através de um índice cujo a numeração começa do zero.

```
>>> x = [1,'Maria',True,120.2]
>>> x[0]
1
>>> x[2]
True
>>> x[1]
'Maria'
>>> x[3]
120.2
```

Acessando elementos de uma lista

- Um item pode ser selecionado individualmente pelo uso dos colchetes, Lista[índice].

```
>>> x = [1,['Maria','Da','Silva'],32,120.2]
>>> x[0]
1
>>> x[1]
['Maria', 'Da', 'Silva']
>>> x[1][0]
'Maria'
>>> x[1][0] + x[1][1] + x[1][2]
'MariaDaSilva'
```

Acessando elementos de uma lista

- Acessar com índices negativos, a partir do -1, percorrem a lista de trás para frente.

```
>>> x = [1,['Maria','Da','Silva'],32,120.2]
>>> x[-1]
120.2
>>> x[-3]
['Maria', 'Da', 'Silva']
>>> x[-3][1]
'Da'
```

Modificando elementos de uma lista

- Como a lista é mutável, os valores nela armazenados podem ser alterados e novos valores podem ser adicionados.

```
>>> x = [1,['Maria','Da','Silva'],32,120.2]
>>> x[2] = ['Pedro','Santos']
>>> x
[1, ['Maria', 'Da', 'Silva'], ['Pedro', 'Santos'], 120.2]
>>> x[1]= 2
>>> x
[1, 2, ['Pedro', 'Santos'], 120.2]
>>> x[3]
120.2
```

Operações em listas

- append() - Adiciona elementos a lista.

```
>>> nomes = ['Yuri','Joao','Maria']
>>> nomes.append('Pedro')
>>> nomes
['Yuri', 'Joao', 'Maria', 'Pedro']
```

- ‘+’ – Concatena Listas

```
>>> nomes = ['Yuri','Joao','Maria']
>>> nomes = nomes + ['Pedro']
>>> nomes
['Yuri', 'Joao', 'Maria', 'Pedro']
```

```
>>> nomes = ['Yuri','Joao','Maria']
>>> nomes = nomes + ['Pedro','Debora']
>>> nomes
['Yuri', 'Joao', 'Maria', 'Pedro', 'Debora']
```

Operações em listas

- insert() - Insere na posição especificada um elemento

```
>>> x = ['a','c','d']
>>> x.insert (1, 'b')
>>> x
['a', 'b', 'c', 'd']
```

- pop() - Remove e retorna o elemento da posição especificada

```
>>> x = ['a','b','c','d']
>>> x.pop(3)
'd'
>>> x
['a', 'b', 'c']
```

Operações em listas

- remove() - Remove o elemento especificado

```
>>> x = ['a','b','c','d']
>>> x.remove('c')
>>> x
['a', 'b', 'd']
```

Operações em listas

- ‘*’ - Multiplica a concatenação da lista

```
>>> nomes = ['Yuri','Joao','Maria'] * 2
>>> nomes
['Yuri', 'Joao', 'Maria', 'Yuri', 'Joao', 'Maria']
```

- clear() – Esvazia a lista

```
>>> x = [1,'Maria',True,120.2]
>>> x.clear()
>>> x
[]
```

Operações em listas

- count() - Retorna quantas ocorrências há do argumento passado

```
>>> x = ['Pedro','Maria','Joao','Maria']
>>> x.count('Maria')
2
```

- index() – Retorna o índice da primeira ocorrência do argumento passado

```
>>> x = ['Pedro','Maria','Joao','Maria']
>>> x.index('Maria')
1
```

Operações em listas

- reverse() - Inverte a ordem dos elementos

```
>>> x = [100,50,20,0]
>>> x.reverse()
>>> x
[0, 20, 50, 100]
```

- sort(reverse = True ou False) – Ordena a lista em ordem crescente ou decrescente.

```
>>> x = [4,19,5,2,99,102]
>>> x.sort(reverse=True)
>>> x
[102, 99, 19, 5, 4, 2]
```

```
>>> x = [4,19,5,2,99,102]
>>> x.sort(reverse=False)
>>> x
[2, 4, 5, 19, 99, 102]
```

Operações em listas

- len() - retorna o tamanho da lista.

```
>>> x = [4,19,5,2,99,102]
>>> len(x)
6
```

- min() e max - retorna o menor/maior valor da lista.

```
>>> x = [4,19,5,2,99,102]
>>> min(x)
2
>>> max(x)
102
```

Fatiamento de listas

- É possível acessar partes da lista por meio de fatiamentos. Para isso, há três parâmetros, separados por dois pontos:

lista[inicio: fim: passo]

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> x[0:2:1]
[1, 2]
>>> x[0:4:2]
[1, 3]
>>> x[4::-1]
[5, 4, 3, 2, 1]
```

O passo negativo, que indica que o fatiamento percorre a lista no sentido inverso

Fatiamento de listas

- É possível também omitir os parâmetros. Por padrão, o início vale 0, o fim vale o índice do último elemento + 1 (tamanho da lista) e o passo vale 1.

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> x[:2]
[1, 2]
>>> x[5:]
[6, 7, 8, 9, 0]
>>> x[::2]
[1, 3, 5, 7, 9]
```

Criação de listas com range ( )

A função range() define um intervalo de valores inteiros. Associada a list(), cria uma lista com os valores do intervalo. A função range() pode ter de 1 a 3 parâmetros:

- range(n) - gera um intervalo de 0 a n-1
- range(i , n) - gera um intervalo de i a n-1
- range(i , n, p) - gera um intervalo de i a n-1 com intervalo p entre os números

Criação de listas com range ( )

- A função range() define um intervalo de valores inteiros. Associada a list(), cria uma lista com os valores do intervalo.

```
>>> lista = list(range(6))
>>> lista
[0, 1, 2, 3, 4, 5]
>>> lista = list(range(2,10))
>>> lista
[2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> lista = list(range(2,10,2))
>>> lista
[2, 4, 6, 8]
```

Tuplas em python

- Tupla, assim como a Lista é um conjunto sequencial de valores, onde cada valor é identificado através de um índice.

- A principal diferença entre elas é que as tuplas são imutáveis, ou seja, seus elementos não podem ser alterados.

- Este fato faz com que as tuplas sejam mais compactas e eficazes em termos de memória e eficiência.

```
nome_tupla = (valor1, valor2, ..., valorN)
```

Tuplas em python

```
T = (1,2,3,4,5)
print(T)
(1, 2, 3, 4, 5)
print(T[3])
4
```

```
T[3] = 8
Traceback (most recent call last):
  File "C:/Python34/teste.py", line 4, in <module>
    T[3] = 8
TypeError: 'tuple' object does not support item assignment
```

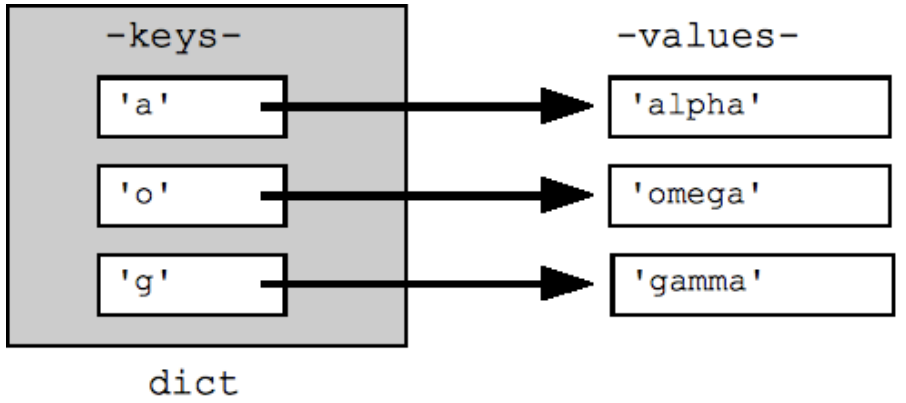
Desempacotamento das tuplas

Uma ferramenta muito utilizada em tuplas é o **desempacotamento** , que permite atribuir os elementos armazenados em uma tupla a diversas variáveis

```
T = (10,20,30,40,50)
a,b,c,d,e = T
print("a=",a,"b=",b)
a= 10 b= 20
print("d+e=",d+e)
d+e= 90
```

Dicionários em python

- Dicionário é um conjunto de valores, onde cada valor é associado a uma chave de acesso.
- Listas e tuplas são indexadas pela ordem dos elementos, isto é, são ordenadas. Já os dicionários, não são.
- Os dicionários, cada elemento recebe uma etiqueta. Além disso, são mutáveis e heterogêneas.



Declaração de Dicionários

Um dicionário em Python é declara do da seguinte forma:

```
nome_dicionario = { chave1 : valor1,
chave2 : valor2,
chave3 : valor3,
chaveN : valorn
}
```

Exemplo de Dicionário

```
D={"arroz": 17.30, "feijão":12.50,"carne":23.90,"alface":3.40}
print(D)
{'arroz': 17.3, 'carne': 23.9, 'alface': 3.4, 'feijão': 12.5}
print(D["carne"])
23.9
```

Operações em Dicionários

- Inserindo e alterando um elemento de um dicionário.

```
>>> inventario = {'bananas':21,'morango':56}
>>> print(inventario)
{'bananas': 21, 'morango': 56}

>>> inventario['laranja']=23
>>> print(inventario)
{'bananas': 21, 'morango': 56, 'laranja': 23}

>>> inventario['laranja']= 78
>>> print(inventario)
{'bananas': 21, 'morango': 56, 'laranja': 78}
```

Operações em Dicionários

- O comando *del* remove um par chave-valor de um dicionário.

```
>>> inventario = {'abacaxis': 430, 'bananas': 312, 'laranjas': 525, 'peras': 217}
>>> print inventario
{'laranjas': 525, 'abacaxis': 430, 'peras': 217, 'bananas': 312}
```

```
>>> del inventario['peras']
>>> print inventario
{'laranjas': 525, 'abacaxis': 430, 'bananas': 312}
```

Operações em Dicionários

- A função *len* também funciona com dicionários; retornando o número de pares chave-valor.

```
>>> inventario = {'abacaxis': 430, 'bananas': 312, 'laranjas': 525, 'peras': 217}
>>> print inventario
{'laranjas': 525, 'abacaxis': 430, 'peras': 217, 'bananas': 312}
```

```
>>> len(inventario)
4
```

Operações em Dicionários

- O método *keys* recebe um dicionário e retorna uma lista com as chaves.

```
>>> ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
>>> print ing2esp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

```
>>> ing2esp.keys()
['one', 'three', 'two']
```

Operações em Dicionários

- O método *values* é parecido; retorna a lista de valores de um dicionário.

```
>>> ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
>>> print ing2esp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

```
>>> ing2esp.values()
['uno', 'tres', 'dos']
```

Operações em Dicionários

- O método *items* retorna os dois, na forma de uma lista de tuplas - cada tupla com um par chave-valor.

```
>>> ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
>>> print ing2esp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

```
>>> ing2esp.items()
[('one','uno'), ('three','tres'), ('two','dos')]
```

Operações em Dicionários

- O operador *in* é um operador lógico que testa se uma chave está no dicionário.

```
>>> ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
>>> print ing2esp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

```
>>> 'one' in ing2esp
True
>>> 'deux' in ing2esp
False
```