

Linguagem de Programação Orientada a Objetos I

Python – Estruturas de Dados

Prof. Tales Bitelo Viegas

<https://fb.com/ProfessorTalesViegas>

Listas

- ▶ Enumeração de dados para melhor organização da informação
- ▶ Um list é uma sequência de dados (da mesma forma que o tipo str (string))
- ▶ Podem conter elementos de tipos diferentes
- ▶ A representação de listas é feita por []
- ▶ São sequências mutáveis

Listas

```
[>>> # Usando o construtor
[... curso = list()
>>> type(curso)
<class 'list'>
>>> # Mais comum (usando [])
[... curso = []
>>> type(curso)
<class 'list'>
>>> len(curso)
0
>>> █
```

Listas

```
[>>> algPares = [0,2,4,6,8]
[>>> print(algPares)
[0, 2, 4, 6, 8]
[>>> socios = ['Tales', 'Felipe', 'Telmo', 'Jones', 'Lara']
[>>> print(socios)
['Tales', 'Felipe', 'Telmo', 'Jones', 'Lara']
[>>> len(socios)
5
>>> ]
```

Listas com tipos variáveis

- ▶ Os elementos da lista não precisam ser do mesmo tipo

```
[>>> curso = ['Tales Viegas', ['Python', 15, 1.0, 'CCOMP – ULBRA']]  
[>>> len(curso)  
2  
[>>> alunos = [['Fabio', 20], ['Pedro', 21], ['Tiago', None]]  
[>>> len(alunos)  
3  
[>>> ]
```

- A list curso possui 2 elementos, um str e uma list
- A list alunos possui 3 elementos, todos list
- Note que o segundo elemento da list curso é outra list, que possui elementos de tipos variados entre str, float e int
- Note também o None na lista alunos, ele é o equivalente ao null do C/Java

Acessos aos valores

```
>>> alunos = [['Fabio',20],['Pedro',21],['Tiago',None]]  
>>> alunos[1]  
['Pedro', 21]  
>>> alunos[-1]  
['Tiago', None]  
>>> alunos[1:3]  
[['Pedro', 21], ['Tiago', None]]  
>>> alunos[::-1]  
[['Tiago', None], ['Pedro', 21], ['Fabio', 20]]  
>>> alunos[1][0]  
'Pedro'  
>>> nome,idade = alunos[0]  
>>> nome  
'Fabio'  
>>> idade  
20  
>>>
```

De strings para listas

- ▶ Uso do construtor para converter de string para lista

```
>>> texto = 'Tales'  
>>> lista = list(texto)  
>>> lista  
['T', 'a', 'l', 'e', 's']
```

- ▶ Cada caractere da string agora será um elemento da list.

Listas podem ser alteradas

- ▶ Elementos das listas podem ser modificados

```
[>>> passaros = ['canário', 'pica-pau', 'papagaio', 'foca']
[>>> passaros[3]
'foca'
[>>> passaros[3] = 'beija-flor'
[>>> passaros
['canario', 'pica-pau', 'papagaio', 'beija-flor']
>>> █
```

- ▶ Diferente das strings, os elementos das lists podem ser alterados

Removendo um elemento

```
>>> texto = 'Teste'  
>>> lTexto = list(texto)  
>>> print(lTexto)  
['T', 'e', 's', 't', 'e']  
>>> del lTexto[3]  
>>> print(lTexto)  
['T', 'e', 's', 'e']  
>>> lTexto.remove('T')  
>>> print(lTexto)  
['e', 's', 'e']  
>>> lTexto.remove('e')  
>>> print(lTexto)  
['s', 'e']  
>>>
```

Para remover todos os elementos

```
>>> ltexto = list('Teste')
>>> while ltexto.count('e'):
...     ltexto.remove('e')
...
>>> ltexto
['T', 's', 't']
>>>
```

Inserção em listas

```
[>>> vogais = ['a']
[>>> vogais.append('i')
[>>> vogais.append('o')
[>>> vogais
['a', 'i', 'o']
[>>> vogais.insert(1,'e')
[>>> vogais
['a', 'e', 'i', 'o']
[>>> vogais.extend(['u'])
[>>> vogais
['a', 'e', 'i', 'o', 'u']
>>> ]
```

Operadores para listas

```
>>> vogais = ['a','e','i','o','u']
>>> 'a' in vogais
True
>>> 'b' in vogais
False
>>> # Concatenacao
... [0,1,2,3,4,5] + [6,7,8,9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> ['a','e','i','o'] + 'u'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list
>>> [1,2,3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Identidade

- ▶ Mesmo listas iguais possuem identidades diferentes

```
>>> lista1 = [1,2,3]
>>> lista2 = [1,2,3]
>>> lista1 == lista2
True
>>> lista1 is lista2
False
>>> [1,2,3] is [1,2,3]
False
>>> [1,2,3] == [1,2,3]
True
>>> id([1,2,3])
4543685448
>>> id([1,2,3])
4543685448
>>> lista1 is lista1
True
```

Referência

- ▶ As listas são passadas por referência (por serem objetos)

```
>>> a = [1,2,3]
>>> b = a
>>> b is a
True
>>> print(b)
[1, 2, 3]
>>> print(a)
[1, 2, 3]
>>> b[1] = 5
>>> print(b)
[1, 5, 3]
>>> print(a)
[1, 5, 3]
```

Comandos

- ▶ Comandos como `min()`, `max()`, `len()` são aplicáveis a sequências, valendo também para listas

```
>>> lis = [0,1,2,3,4,5,6,7,8,9]
>>> len(lis)
10
>>> max(lis)
9
>>> min(lis)
0
```

Filtros para listas (compressões)

- ▶ As listas podem ser filtradas através de compressões
 - [`<exp> for <elemento> in <lista> if <filtro>`]

```
>>> nome = 'Tales'  
>>> # Usando sem o if, apenas para mapear  
... [el for el in nome]  
['T', 'a', 'l', 'e', 's']  
>>> # Usando o if para verificar se eh vogal  
... [letra for letra in nome if letra in 'aeiou']  
['a', 'e']
```

Tuplas

- ▶ As tuplas, assim como as listas, são sequências, no entanto não são mutáveis
- ▶ Podem conter elementos de diferentes tipos dentro delas
- ▶ É possível realizar slices, como em strings e listas, gerando uma nova tupla
- ▶ As tuplas são representadas por ()
- ▶ As tuplas não possuem métodos
- ▶ Não é possível adicionar, remover ou atualizar elementos em uma tupla

Tuplas – Por que usar?

- ▶ São mais rápidas que as listas
- ▶ São usadas na formatação de strings
- ▶ Para “listas” de elementos constantes, é a melhor opção
- ▶ São usadas como parâmetros de funções de parâmetros variáveis
- ▶ É uma sequência de tipos variados que podem ser usados como chave de dicionário

Criando uma tupla

```
>>> minhaTupla = tuple()  
>>> type(minhaTupla)  
<class 'tuple'>  
>>> minhaTupla = (1,2,3)  
>>> type(minhaTupla)  
<class 'tuple'>  
>>> minhaTupla = 1,2,3  
>>> type(minhaTupla)  
<class 'tuple'>  
>>> tuplaUnit = (1)  
>>> type(tuplaUnit)  
<class 'int'>  
>>> tuplaUnit = (1,)  
>>> type(tuplaUnit)  
<class 'tuple'>  
>>> 3 * (15 + 1)  
48  
>>> 3 * (15 + 1,  
(16, 16, 16)
```

Operadores de Tuplas

```
>>> vogais = ('a', 'e', 'i', 'o')
>>> id(vogais)
18444768
>>> vogais = vogais + ('u',)
>>> vogais
('a', 'e', 'i', 'o', 'u')
>>> id(vogais)
12266400
>>> (10, 3) * 4
(10, 3, 10, 3, 10, 3, 10, 3)
```

Funções Retornando Tuplas

- ▶ Tuplas podem servir de retorno para funções. Um exemplo disto é comando divmod(), que retorna o valor do quociente e do resto em uma tupla

```
>>> quo,resto = divmod(30,7)
>>> quo
4
>>> resto
2
```

Dicionários

- ▶ Dicionários são estruturas utilizando um par de chave e valor
- ▶ São representados em Python por { }
- ▶ Os dicionários são mutáveis
- ▶ Não existe o conceito de ordenação em um dicionário

Criação de Dicionários

```
>>> legenda = {'+': 'Soma', '-': 'Subtração', '*': 'Multiplicação', '/': 'Divisão'}
>>> dic = {}
>>> dic = dict()
>>> type(legenda)
<type 'dict'>
>>> type(dic)
<type 'dict'>
>>> dic = dict(nome='Fábio', idade=21)
>>> dic
{'idade': 21, 'nome': 'F\xelbio'}
```

Acessando o valor

- ▶ Acesso ao valor sempre utilizando a chave

```
>>> legenda = {'+': 'Soma', '-': 'Subtração', '*': 'Multiplicação',
...     '/': 'Divisão'}
>>> legenda['+']
'Soma'
>>> legenda['*']
'Multiplica\xe7\xe3o'
```

Modificando um valor

- ▶ Dicionários possuem valores que podem ser modificados através do acesso pelas suas respectivas chaves

```
>>> legenda = {'+': 'Soma', '-': 'Subtração', '*': 'Multiplicação', '/': 'Divisão'}
>>> legenda['+'] = 'Adição'
>>> legenda
{'+': 'Adição', '*': 'Multiplicação', '-': 'Subtração', '/': 'Divisão'}
```

Removendo Valores

► Métodos pop() e popitem()

```
>>> morse = {'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..'}
>>> cod = morse.pop('B')
>>> print(cod)
'...'
>>> morse
{'A': '.-', 'C': '-.-.', 'D': '-..'}
>>> letra, cod = morse.popitem()
>>> cod
'..'
>>> letra
'D'
>>> cod
'..'
```

Removendo Valores

- ▶ Removendo pelo comando del

```
>>> dic = {'um':1, 'dois':2, 'tres':3}
>>> dic
{'um': 1, 'tres': 3, 'dois': 2}
>>> del dic['tres']
>>> dic
{'um': 1, 'dois': 2}
```

Métodos para trabalhar com chaves

```
>>> morse = {'A':'.-','B':'-...','C':'-.-.','D':'-...'}
>>> 'A' in morse
True
>>> 'Z' in morse
False
>>> morse.keys()
dict_keys(['A', 'B', 'C', 'D'])
>>> inChaves = morse.keys()
>>> for chave in inChaves:
...     print(chave)
...
A
B
C
D
```

Métodos para trabalhar com valores

```
>>> morse = {'A':'.-','B':'-...','C':'-.-. ','D':'-..'}  
>>> morse.values()  
dict_values(['.-', '-...', '-.-.', '-..'])  
>>> inValor = morse.values()  
>>> for v in inValor:  
...     print(v)  
  
...  
. -  
- ...  
- . -.  
- .. _
```

Adicionando mais itens

- ▶ Para adicionar mais elementos em um dicionário utiliza-se o método update

```
>>> morse = {'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..'}  
>>> morse.update({'E': '.', 'F': '...-'})  
>>> morse  
{'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '...-'}
```

- ▶ O método update recebe como parâmetro um dicionário e faz a atualização caso já exista a chave, senão faz a adição de um novo item

Dict e Strings

- ▶ Os dicionários, assim como as tuplas, podem ser usados em strings formatadas, só que de forma nomeada

```
>>> siglas = { 'CE':'Ceara', 'PI':'Piaui'}
>>> print 'O %(CE)s tem fronteiras com o %(PI)s' % siglas
O Ceara tem fronteiras com o Piaui
```

```
>>> page = "<html>\n<head><title>% (title)s</title></head>\n<bo
dy>\n\t<h1>% (title)s</h1>\n\t<p>% (texto)s</p>\n</body>\n</html
>"
>>> home = { 'title':'Home', 'texto':'Seja Bem vindo(a) !'}
>>> print page % home
<html>
<head><title>Home</title></head>
<body>
    <h1>Home</h1>
    <p>Seja Bem vindo(a) !</p>
</body>
</html>
```

Conjuntos

- ▶ Conjuntos são tipos de dados que não possuem item repetidos
- ▶ Existem 2 tipos:
 - set() – mutável (como as listas)
 - frozenset() – imutável (como as tuplas)

Métodos

```
>>> #união
>>> pares.union(impares)
set([1, 2, 3, 4, 5, 6])
>>> #ou usando o operador |
>>> impares | pares
set([1, 2, 3, 4, 5, 6])
>>> #Interseção
>>> pares.intersection(impares)
set([])
>>> #ou usando o operador &
>>> pares & impares
set([])
>>> #Diferença
>>> pares.difference(impares)
set([2, 4, 6])
>>> #ou usando o operador -
>>> impares - pares
set([1, 3, 5])
```

Funções

► Sintaxe básica

```
def nome(parametros):  
    <bloco de comandos>
```

Funções

```
>>> def menu():
        print """
Escolha um item:
1) Comprar passagem
2) Verificar destino
3) Sair
"""
>>> menu()
Escolha um item:
1) Comprar passagem
2) Verificar destino
3) Sair
>>> def triplo(x):
        return 3*x
>>> print triplo(4)
12
>>> def curso(nome = "Python"):
        print "Curso de " + nome
>>> curso()
Curso de Python
>>> curso('C#')
Curso de C#
```

Funções

- ▶ Funções também são objetos

```
>>> def divmodF(x,y):  
        return x / y,x % y  
  
>>> divmodF(10,3)  
(3, 1)  
>>> divmodF  
<function divmodF at 0x00BB19B0>  
>>> d = divmodF  
>>> d(10,3)  
(3, 1)
```

Ordem dos parâmetros

- ▶ A ordem dos parâmetros não importa, se forem chamados de forma nomeada

```
>>> def repete(string = "",vezes = 0):  
        print string * vezes  
  
>>> repete('a',10)  
aaaaaaaaaaa  
>>> repete(vezes = 10,string = 'a')  
aaaaaaaaaaa
```

Usando tupla como argumento

```
>>> def teste(*meuparm):
    print type(meuparm)

>>> teste(1,2,3,4)
<type 'tuple'>
>>> teste('a',12,[1,2,3])
<type 'tuple'>
>>> #Note que meuparm é um tupla
>>> def soma(*val):
    s = 0
    for i in val:
        s += i
    return s

>>> soma(1,2,3,4)
10
>>> l = [1,2,3,4]
>>> soma(l)
TypeError: unsupported operand type(s) for +=: 'int' and 'list'
>>> soma(*l)
10
```

Usando dict como argumento

```
>>> def teste2(**args):
    print type(args)

>>> teste2(oi=1,hurra=2,seila='lalala')
<type 'dict'>
>>> fichaTec('Fábio','Pedro',Heroi='Raimundo',Mocinha='Maria')
Diretor: Fábio
Autor: Pedro
Mocinha: Maria
Heroi: Raimundo
>>> def fichaTec(diretor = "",autor = "",**elenco):
    print "Diretor: " + diretor
    print "Autor: " + autor
    print "Elenco:\n" + "-" * 20
    for pers,ator in elenco.iteritems():
        print "%s: %s" % (pers,ator)

>>> fichaTec('Fábio','Pedro',Heroi='Raimundo',Mocinha='Maria')
Diretor: Fábio
Autor: Pedro
Elenco:
-----
Mocinha: Maria
Heroi: Raimundo
```

Funções dentro de Funções

```
>>> def somafixo(x):  
        def valor(y):  
            return x+y
```

```
>>>  
>>> def somafixo(x):  
        def valor(y):  
            return x+y  
        return valor
```

```
>>> com3 = somafixo(3)  
>>> com3(10)  
13  
>>> com5 = somafixo(5)  
>>> com5(17)  
22
```