Laboratório de Programação

Introdução ao Python

Armando Soares Sousa

Departamento de Computação - UFPI

armando@ufpi.edu.br

Listas

As listas são usadas para armazenar vários itens em uma única variável.

As listas são criadas usando colchetes:

L = [item1, item2, …, itemN]

A ordem da lista é definida na declaração dos itens.

Os itens da lista são indexados, o primeiro item tem índice [0], o segundo item tem índice [1] etc.

Uma lista pode ser modificada depois que ela é criada.

A lista permite itens de tipos diferentes

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]

print(thislist[0])
print(thislist[1])
print(thislist[2])
```

```
apple
banana
cherry
```

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]

print(thislist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

Listas

A lista permite itens de tipos diferentes

A lista permite valores duplicados

O tamanho da lista é dado via a *função builtin* len()

L = ["a", "b", "c"]
len(L) = 3

type(L) retorna class 'list'

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

list4 = ["apple", 1, True]
list5 = [list1, list2]

print(list1)
print(list2)
print(list3)
print(list4)
print(list5)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
['apple', 1, True]
[['apple', 'banana', 'cherry'], [1, 5, 7, 9, 3]]
```

Listas

O construtor list()

Também é possível usar o construtor list() ao criar uma nova lista.

```
# Cria uma lista vazia
my_list = list()
print(my_list)

# Cria uma lista com 3 elementos
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

```
[]
['apple', 'banana', 'cherry']
```

Acessando itens de uma lista

Acessando pelo índice do elemento: 0, 1, 2, 3, 4…

Uso de índices negativos: -1, -2, -3, -4, -5

A indexação negativa significa começar do fim

-1 refere-se ao último item, -2 refere-se ao penúltimo item, etc.

Faixa de índices: 2..5, ..4

O limite superior do range possui intervalo aberto

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
cherry
```

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,
#and note that the item in position 5 is NOT included
# O limite superior do range tem intervalo ABERTO
```

```
['cherry', 'orange', 'kiwi']
```

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])

#This will return the items from index 0 to index 4.

#Remember that index 0 is the first item, and index 4 is the fifth item
#Remember that the item in index 4 is NOT included
```

```
['apple', 'banana', 'cherry', 'orange']
```

Acessando itens de uma lista

Checando se um item existe na lista

Você pode usar o operador **in** ou **not in**

if elemento1 **in** L:
    O elemento está na lista
else:
    O elemento não está na lista

if elemento1 **not in** L:
    O elemento não está na lista
else:
    O elemento está na lista

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")

if "orange" in thislist:
  print("Yes, 'orange' is in the fruits list")
else:
  print("No, 'orange' is not in the fruit list")
```

```
Yes, 'apple' is in the fruits list
No, 'orange' is not in the fruit list
```

Acessando itens de uma lista

Alterando elementos

Adicionando elementos

- append() - insere ao final
- insert() - insere de acordo com o índice

Removendo elementos

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "orange"

print(thislist)
```

```
['apple', 'orange', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
thislist = ["apple", "banana", "cherry"]
# indice 0 - posicao 1
# indice 1 - posicao 2
# indice 3 - posicao 3

# insere na posicao 2
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

Acessando itens de uma lista

Removendo elementos

O método remove() remove o item especificado.

O método pop() remove o índice especificado.

Se você não especificar o índice, o método pop() remove o último item.

A palavra-chave **del** também remove o índice especificado

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
['apple', 'cherry']
```

Loop em Listas

Você pode percorrer os itens da lista usando um loop **for .. in**

Você também pode percorrer os itens da lista referindo-se ao seu número de índice.

Use as funções **range()** e **len()** para criar um iterável adequado.

```
thislist = ["apple", "banana", "cherry"]
for fruta in thislist:
  print(fruta)
```

```
apple
banana
cherry
```

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(f'indice: {i}, {thislist[i]}')
```

```
indice: 0, apple
indice: 1, banana
indice: 2, cherry
```

Ordenando uma lista

Os objetos de lista têm um método **sort()** que classificará a lista alfanumericamente, em ordem crescente, por padrão.

Para classificar de forma descendente, use o argumento de palavra-chave **reverse = True**

Obs: por padrão o método sort() é case sensitive

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

```
['Kiwi', 'Orange', 'banana', 'cherry']
```

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

```
[23, 50, 65, 82, 100]
```

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

```
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

Concatenando duas listas

Existem várias maneiras de juntar ou concatenar duas ou mais listas em Python.

Uma das maneiras mais fáceis é usar o operador +.

Outra maneira de juntar duas listas é anexando todos os itens da lista2 na lista1, um por um.

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Métodos mais comuns de uma Lista

Python tem um conjunto de métodos integrados que você pode usar em listas.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Tuplas

Um **Tupla** é uma estrutura muito parecida com Listas, a principal diferença é uma estrutura que não permite alterar o conteúdo dos seus elementos.

Uma tupla é uma coleção que pode ser ordenada, mas seu conteúdo é imutável.

Tuplas são escritas com colchetes.

minha_tupla = (item1, item2, ..., itemN)

Permite elementos duplicados

Os elementos podem ter tipos de dados diferentes

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Acesso a elementos de Tuplas

O acesso dos elementos é feito informando o índice do elemento.

O índice inicia com 0

Obs: Não é possível alterar elementos de uma tupla.

Não é possível adicionar ou remover elementos de uma tupla.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Packing and Unpacking de uma tupla

Quando criamos uma tupla, normalmente atribuímos valores a ela. Isso é chamado de "empacotar" uma tupla.

```
# Packing
fruits = ("apple", "banana", "cherry")

# Unpacking
(x, y, z) = fruits

print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

Mas, em Python, também podemos extrair os valores de volta para as variáveis. Isso é chamado de "desempacotar":

Percorrer uma Tupla usando Loops

Você pode percorrer os itens da tupla usando um loop **for**..**in**

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

Você também pode percorrer os itens da tupla referindo-se ao seu número de índice.

Use as funções **range()** e **len()** para criar um iterável adequado.

```
thistuple = ("apple", "banana", "cherry")

for i in range(len(thistuple)):
  print(thistuple[i])
```

```
apple
banana
cherry
```

Concatenando tuplas

Para juntar duas ou mais tuplas você pode usar o operador +:

Neste caso, como uma tupla é imutável, é preciso criar uma terceira tupla para receber o resultado da concatenação da tupla1 e tupla2.

tupla3 = tupla1 + tupla2

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Métodos mais comuns de Tuplas

Python tem dois métodos integrados que você pode usar em tuplas.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

```
thistuple = (1, 3, 5, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(f"O número 5 aparece {x} vezes")
```

```
O número 5 aparece 3 vezes
```

```
thistuple = (1, 3, 5, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(f"O número 8 aparece na posição {x}")
```

```
O número 8 aparece na posição 4
```

Sets

Os conjuntos são usados para armazenar vários itens em uma única variável.

Um conjunto é uma coleção não ordenada, imutável* e não indexada.

* Nota: Os itens definidos não podem ser alterados, mas você pode remover itens e adicionar novos itens.

Conjuntos são declarados com chaves {..}.

meu_conjunto = {item1, item2, …, itemN}

onde item1 != item2, item1 != itemN, item2 != ItemN

Sets

A lista de elementos de um conjunto não é ordenada pela sua declaração, ou seja, os itens permanecerão os mesmos, mas a ordem de cada ítem é randômica.

Conjuntos não permitem itens duplicados.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

# Note: the set list is unordered, meaning: the items will appear in a random order.

# Refresh this page to see the change in the result.
```

```
{'cherry', 'banana', 'apple'}
```

Acessando itens de Sets

Você não pode acessar itens em um conjunto referindo-se a um índice ou chave.

Mas você pode percorrer os itens do conjunto usando um loop for ou perguntar se um valor especificado está presente em um conjunto usando a palavra-chave in.

Depois que um conjunto é criado, você não pode alterar seus itens, mas pode adicionar novos itens.

É possível checar se um item pertence ao conjunto usando o comando **in**

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
apple
banana
cherry
```

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

```
thisset = {"apple", "banana", "cherry"}

if "banana" in thisset:
    print("banana está em thisset" )
```

```
banana está em thisset
```

Adicionando e removendo itens de um Set

Depois que um conjunto é criado, você não pode alterar seus itens, mas pode adicionar novos itens ou remover itens existentes.

Para adicionar um item a um conjunto, use o método add().

Para remover um item de um conjunto, use o método remove().

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'orange', 'apple', 'cherry', 'banana'}
```

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

Percorrendo (Loop) os itens de um Set

Você pode percorrer os itens definidos usando um loop **for .. in**

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
cherry
apple
banana
```

Concatenando dois conjuntos

Você pode usar o método **union()** que retorna um novo conjunto contendo todos os itens de ambos os conjuntos

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 'a', 'b', 2, 3, 'c'}
```

```
set1 = {"a", "b" , "c"}
set2 = {"a", "b" , "c", "d", "e"}

set3 = set1.union(set2)
print(set3)
```

```
{'c', 'b', 'e', 'a', 'd'}
```

Métodos mais comuns para um Set

Python tem um conjunto de métodos integrados que você pode usar em conjuntos.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others