

CC8210 – NCA210

Programação Avançada I

Prof. Reinaldo A. C. Bianchi

Prof. Isaac Jesus da Silva

Prof. Danilo H. Perico

Conteúdo Programático

AULA	DATA	TEORIA
1	10/08/20	Introdução a disciplina. Introdução a Python :Estrutura Sequencial, Condicional e Estruturas de Repetição;
2	17/08/20	Vetores ou Listas;
3	24/08/20	Listas Aninhadas ou Matrizes;
4	31/08/20	Modularização (Funções com/sem passagem de parâmetro, com/sem retorno);
5	14/09/20	Manipulação de Strings
6	21/09/20	Manipulação de Arquivos
7	28/09/20	Dicionários e Tuplas
	05/10/20	P1
8	19/10/20	Introdução à POO - Classes e Objetos
9	26/10/20	Introdução à biblioteca Pandas
10	09/11/20	Introdução à bibliotecas matemáticas e gráficas numpy e matplotlib. Manipulação de gráficos múltiplos e de barras;
11	16/11/20	Interface Gráfica do Usuário (GUI)
12	23/11/20	P2

Relembrando a aula passada: Python

Python

- É uma Linguagem de Programação.
- Linguagens de programação são usadas como um meio de comunicação entre os computadores e os humanos.
- Codificam os algoritmos para uma linguagem que o computador pode entender.
- Línguagem de alto nível.
- Interpretada.

Por que usar Python???

- Vantagens:
 - Fácil de programar e aprender a programar;
 - É portável a quase todos os sistemas operacionais;
 - Rápida prototipagem;
 - Pode fazer integração com outras linguagens;
 - Produtividade.

Como programar em Python?

- Você pode baixar o programa Python no site:
<https://www.python.org>
- Ou baixar o Python já com vários pacotes no site:
<https://www.anaconda.com>
- Ou pode usar o Python no site:
<https://www.repl.it>

Spyder (Python 3.8)

/Users/rbianchi

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py

exemplo01.py*

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 nome = input ("Entre com o seu nome: ")
print(nome)
10
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]:
```

LSP Python: ready conda: base / Python 3.8.3 Line 9, Col 20 UTF-8 LF RW Mem 59%

Python - Primeiros Programas - Saída de dados

- Saída de dados:

```
print("Olá mundo!")
```

- A função print informa que vamos exibir algo na tela.

Python - Primeiros Programas - Variáveis

- Variáveis são usadas para criar uma região de memória em que vamos armazenar dados:

```
nome = input("Entre com o seu nome:")
```

- nome é uma variável que vai armazenar a entrada que o usuário digitar

Python - Primeiros Programas - Exemplo

- Código:

```
1 nome = input("Entre com o seu nome: ")  
2 print(nome)
```

Variáveis - Tipos de Dados

- Para descobrir qual é o tipo de determinado dado, podemos utilizar a função `type()` no Python.

```
In [1]: type(3.6)
```

```
Out[1]: float
```

```
In [2]: type(2)
```

```
Out[2]: int
```

```
In [3]: type("olá")
```

```
Out[3]: str
```

```
In [4]: type("2.8")
```

```
Out[4]: str
```

```
In [5]: type(2+6j)
```

```
Out[5]: complex
```

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Entre com o seu nome: Reinaldo Bianchi
Reinaldo Bianchi

In [2]: type (3)
Out[2]: int

In [3]: type (nome)
Out[3]: str

In [4]: type (3.1415926535897932384626433)
Out[4]: float

In [5]:

Comando **if**

- Em Python, e em várias outras linguagens de programação, o comando principal para a realização de decisões é o **if**
- Sintaxe do **if** no Python:

```
if <condição>:  
    bloco verdadeiro
```
- **If** nada mais é do que nosso **se**
 - Em português, podemos entender o comando if da seguinte forma:
 - **Se a condição for verdadeira, faça alguma coisa**

Comando **if** - indentação

- O bloco que será executado se a condição do if for verdadeira fica **indentado** com relação ao comando if
- **Indentação** é o **rekuo** (deslocamento do texto à direita)
- **Indentação:** neologismo derivado da palavra em inglês *indentation*
- ***BLOCOS SÃO DEFINIDOS PELA IDENTAÇÃO!!!***

Comando **else**

- O comando **else** (senão) é utilizado nos casos em que a segunda condição é simplesmente o **contrário** da primeira.
- Sempre utilizado como uma sequência de um **if**
- Sintaxe:

```
if <condição>:  
    bloco verdadeiro  
else:  
    bloco contrário
```

Comando **elif**

- O comando **elif** (*else if*- senão se) substitui, em muitos casos, a necessidade do aninhamento
- É sempre utilizado como uma sequência de um **if**
- Sintaxe:

```
if <condição 1>:  
    #bloco se a condição 1 for verdadeira  
elif <condição 2>:  
    #bloco se a condição 1 for falsa e a condição 2 verdadeira  
else:  
    #bloco contrário a todas outras condições
```

Comentários

- São textos que não são interpretados como código de programa
- Servem para documentar o programa
- No Python, uma linha de comentário começa com o símbolo # (padrão mais comum)
- Exemplo:

```
a = 8
#isso é um comentário
print(a)
```

Comando **while**

- O comando **while** (enquanto) serve para executarmos alguma repetição **enquanto** uma condição for verdadeira (True)
- Sintaxe:

```
while <condição>:  
    #bloco que será repetido enquanto a condição for verdadeira
```

while infinito

```
while True:
```

*#bloco que sempre será executado,
#nunca sai do loop de repetição*

Comando **break**

- Porém, mesmo quando utilizamos um **while** infinito, é possível que em determinadas situações o programa precise sair do loop de repetição.
- Esta interrupção pode ser alcançada com o comando **break**
- O comando **break** pode ser utilizado para interromper o **while**, independentemente da condição

Comando for

- for é a estrutura de repetição mais utilizada
- Sintaxe:

```
for <referência> in <sequência>:  
    #bloco de código que será repetido  
    #a cada iteração
```
- Durante a execução, a cada iteração, a referência aponta para um elemento da sequência.
- Uma vantagem do for com relação ao while é que o contador não precisa ser explícito!

Comando **for** - Exemplo

- Calcular a somatória dos números de 0 a 99

```
somatoria = 0  
  
for x in range(0,100):  
    somatoria = somatoria + x  
print(somatoria)  
  
4950
```

A função **range(*i, f, p*)** é bastante utilizada nos laços com **for**

Ela gera um conjunto de valores inteiros:

- Começando de *i*
- Até valores menores que *f*
- Com passo *p*

Se o passo *p* não for definido, o padrão de 1 será utilizado.

For versus While

- *for* loops are traditionally used when you have a block of code which you want to repeat a fixed number of times.
- The Python *for* statement iterates over the members of a sequence in order, executing the block each time.
- Contrast the *for* statement with the "while" loop, used when a condition needs to be checked each iteration, or to repeat a block of code forever.

Vetores ou Listas

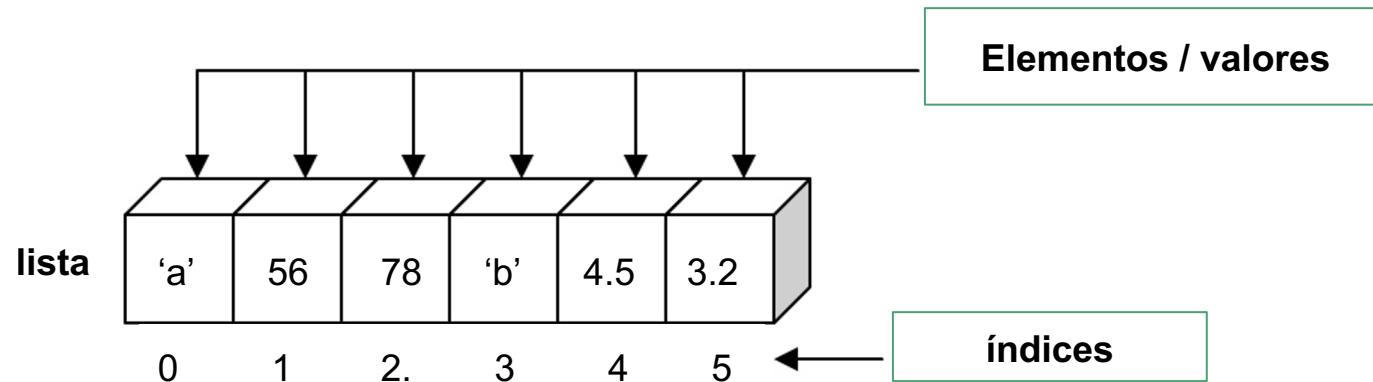
Listas ou Vetores

Listas

Lista

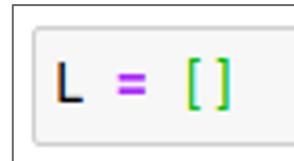
- Uma lista é uma variável que armazena um conjunto de valores.
- Lista é um tipo de variável que permite o armazenamento de valores com tipos homogêneos ou heterogêneos:
 - Homogêneos = do mesmo tipo.
 - Heterogêneo = de tipos diferentes.
- Os valores armazenados em uma lista são acessados por um índice.

Lista heterogênea



Lista

- Para indicar que uma variável é uma lista, o símbolo de colchetes [] é utilizado para delimitar o conjunto
- Sintaxe - criando uma lista chamada L:



- L é uma lista vazia

Lista - Exemplo

- Criando uma lista chamada z com 3 números inteiros

```
z = [5, 7, 1]
print(z)

[5, 7, 1]
```

- Dizemos que z tem tamanho 3

Lista versus Vetores

- **Lists are ordered:**
 - The items in the list appear in a specific order. This enables us to use an index to access to any item.
- **Lists are mutable:**
 - You can add or remove items after a list's creation.
- **List elements do not need to be unique.**
 - Item duplication is possible, as each element has its own distinct place and can be accessed separately through the index.
- **Elements can be of different data types.**

Lista - Acesso aos elementos

- Exemplo:

```
z = [5, 7, 1]
print(z)

[5, 7, 1]
```

- Para acessarmos o primeiro número da lista z, utilizamos a notação: z[0]
- Ou seja, da lista z queremos pegar o valor armazenado no índice 0.

```
z = [5, 7, 1]
print(z[0])
print(z[1])
print(z[2])

5
7
1
```

Lista

- Utilizando o nome de uma lista com o índice desejado, podemos também modificar o conteúdo armazenado.
- Exemplo: Alterando o valor do primeiro elemento da lista

z

```
z = [5, 7, 1]
z[0] = 32
print(z)

[32, 7, 1]
```

Lista - Fatiamento (Slicing)

- No Python, podemos também fatiar as listas
- Ou seja, pegar somente partes de uma lista
- Exemplo:

```
p = [1,2,3,4,5,6]  
print(p[0:5])
```

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

```
print(p[:4])
```

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

```
print(p[1:3])
```

```
[2, 3]
```

```
print(p[-1])
```

```
6
```

Lista - Adicionando elementos no fim da lista

- Podemos ainda adicionar novos elementos no fim da lista
- Para isto, utilizamos o método ***append(item)***
- Exemplo:

```
z = [32, 7, 1]
print(z)

[32, 7, 1]

z.append("oi")
print(z)

[32, 7, 1, 'oi']
```

Lista - Adicionando elemento em qualquer lugar

- Podemos ainda adicionar novos elementos em qualquer lugar da lista
- Para isto, utilizamos o método *insert(índice, item)*
- Exemplo:

```
z = [32, 7, 1]
print(z)
[32, 7, 1]

z.insert(1,"oi")
print(z)
[32, 'oi', 7, 1]
```

Lista - Adicionando diversos elementos

- Podemos ainda adicionar diversos elementos ao mesmo tempo no final da lista
- Para isto, utilizamos o método ***extend(iterable)***
- Exemplo:

```
a1 = [1, 2]
a2 = [1, 2]
b = (3, 4)
```

```
a1.extend(b)
print(a1)
```

```
a2.append(b)
print(a2)
```

RESULTADO:
[1, 2, 3, 4]
[1, 2, (3, 4)]

Lista - Removendo da lista pelo índice

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *pop(índice)*
- Exemplo:
- Se nenhum índice for especificado, ele retorna o ultimo elemento.

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

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

```
z.pop(1)  
print(z)
```

```
['a', 'c', 'd', 'e']
```

Lista - Removendo da lista pelo elemento

- Podemos remover um elemento da lista
- Para isto, utilizamos o método ***remove(item)***
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

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

```
z.remove("d")  
print(z)
```

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

```
z = [1,2,3,1,4,5,1]  
z.remove(1)  
print(z)
```

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

Lista - Cópia

- A cópia de uma lista para uma nova variável requer alguma atenção!
- Por exemplo, se quisermos copiar a lista z para uma nova variável chamada z1, o mais natural seria o seguinte:
 - $z1 = z$
- Porém, quando fazemos isso no Python, criamos duas variáveis que referenciam a mesma lista!
- É como se dêssemos dois nomes para a mesma lista...

Lista - Cópia

- Exemplo:
- Quando alteramos o elemento na lista *z*, a alteração ocorre também na lista *z1*

```
z = [4,5,3,6]
z1 = z
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

```
antes da alteração na lista z
[4, 5, 3, 6]
[4, 5, 3, 6]
depois da alteração na lista z
[4, 98, 3, 6]
[4, 98, 3, 6]
```

Lista - Cópia

- Para criarmos uma cópia independente, utilizamos a sintaxe:

```
z1 = z[ : ]
```

- Outra opção é utilizar o método `list.copy()`.
 - Return a shallow copy of the list.
 - Equivalent to `a[:]`.

```
z = [4,5,3,6]
z1 = z[:]
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

antes da alteração na lista z
[4, 5, 3, 6]
[4, 5, 3, 6]
depois da alteração na lista z
[4, 98, 3, 6]
[4, 5, 3, 6]

Lista - Tamanho da lista

- Como temos os métodos para incluir e remover dados das listas, nem sempre sabemos qual é o tamanho exato que a lista tem
- Para descobrirmos o tamanho da lista, utilizamos o método ***len(lista)***
- Exemplo:

```
a = [3, 4, 5]
print(len(a))

a.append(9)
a.append(11)
print(len(a))
```

```
3
5
```

Lista - Pesquisando na lista

- Podemos pesquisar se um elemento está na lista
- Para isso, verificamos do primeiro ao último comparando com o que queremos encontrar.
- Para percorrer listas, utilizamos uma estrutura de repetição: while ou for
- A estrutura for é otimizada para trabalhar com listas

Lista - Pesquisando na lista

- Se a ideia é somente falar se o elemento está ou não na lista, podemos utilizar uma estrutura simples:

```
z = ["a", "b", "c", "d", "e"]
if "c" in z:
    print("Encontrado!")
else:
    print("Não encontrado!")
```

Encontrado!

Lista - Pesquisando na lista

- Ou podemos procurar elemento a elemento.
- Exemplo: Procurar o elemento “c” na lista z

```
z = ["a", "b", "c", "d", "e"]
for elemento in z:
    if elemento == "c":
        print("Elemento encontrado!")
        break
else:
    print("Elemento não encontrado!")
```

Elemento encontrado!

Lista - Pesquisando na lista

- Contudo, nem sempre encontrar o elemento é suficiente.
- Muitas vezes, precisamos saber qual é a sua posição na lista.
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]
for indice in range(len(z)):
    if z[indice] == "c":
        print("Elemento encontrado no índice %d" % indice)
        break
    else:
        print("Elemento não encontrado!")
```

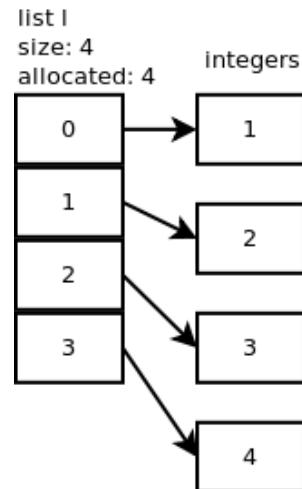
Elemento encontrado no índice 2

Listas – outros métodos úteis

- `list.clear()`
 - Remove all items from the list. Equivalent to `del a[:]`.
- `list.count(x)`
 - Return the number of times *x* appears in the list.
- `list.sort(key=None, reverse=False)`
 - Sort the items of the list in place.
- `list.reverse()`
 - Reverse the elements of the list in place.

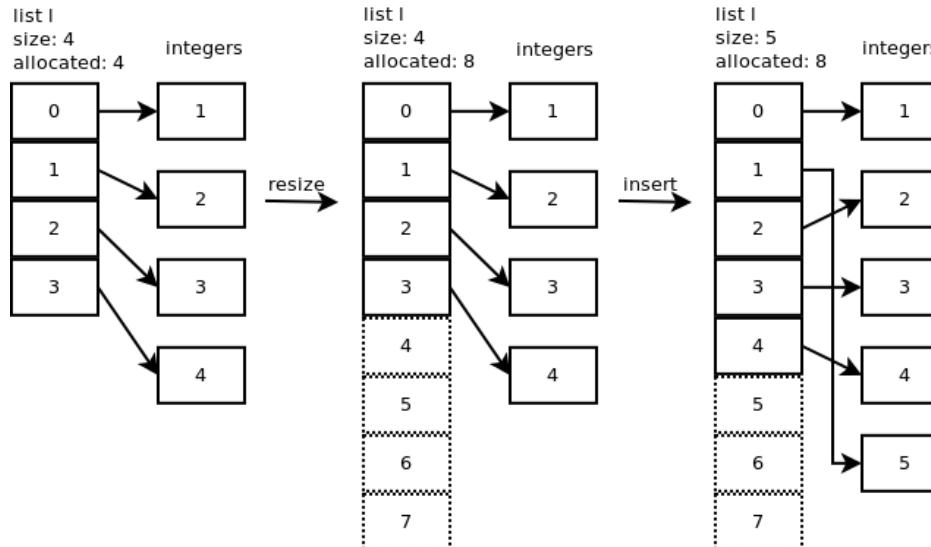
Listas, mais a fundo

- Em Python a lista é implementada como um vetor de ponteiros que aponta cada elemento armazenado:



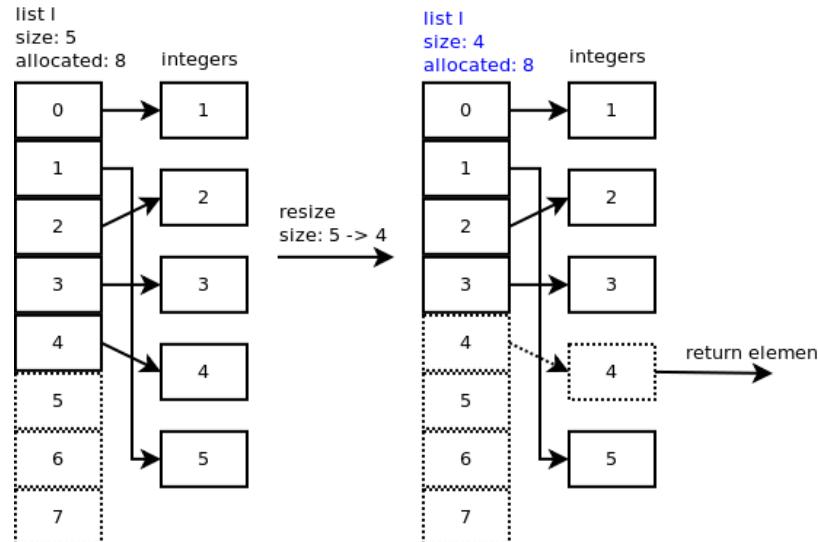
Listas, mais a fundo

- Isto permite a inserção de elementos em qualquer posição: `l.insert(1,5)`



Listas, mais a fundo

- Isto permite a remoção de elementos em qualquer posição: `l.pop()`



Vetores ou Arrays

- Além das listas, existem os vetores, ou arrays em inglês.
- An array is also a data structure that stores a collection of items.
 - Like lists, arrays are ordered, mutable, enclosed in square brackets, and able to store non-unique items.
- The Python array module requires all array elements to be of the same type.
 - To create an array, you'll need to specify a value type.

Vetores ou Arrays

- To use arrays in Python, you need to import either an array module:

```
import array as arr
```

- The code below creates na array of integers:

```
array_1 = arr.array("i", [3, 6, 9, 12])
print(array_1)
print(type(array_1))
```

Arrays versus Lists

- When should you use a list and when use an array?
- If you need to store a relatively short sequence of items and you don't plan to do any mathematical operations with it, a list is the preferred choice.
 - This data structure will allow you to store an ordered, mutable, and indexed sequence of items without importing any additional modules or packages.

Arrays versus Lists

- When should you use a list and when use an array?
- If you have a very long sequence of items, consider using an array.
 - This structure offers more efficient data storage.
- If you plan to do any numerical operations with your combination of items, use an array.
 - Data analytics and data science rely heavily on arrays.
- Veremos melhor os arrays quando usarmos NumPy.

Conclusão

- Listas são usadas como estruturas de dados lineares.
- Deixe os arrays para a matemática, que será vista mais a frente no curso.

Fim

```
graph TD; list[list] --> remove[list.remove()]; list --> append[list.append()]; list --> reverse[list.reverse()]; list --> sort[list.sort()]; list --> sorted[sorted()]; list --> copy[list.copy()]; list --> pop[list.pop()]; list --> insert[list.insert()]; list --> index[IndexError]
```