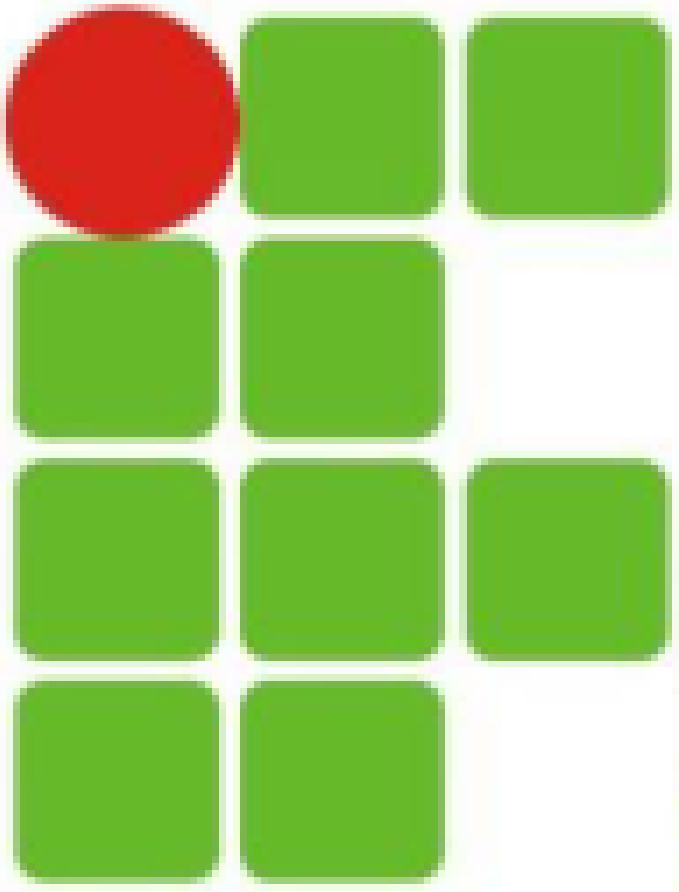
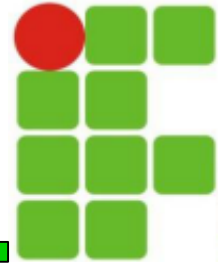


Python:
Listas

Galileu Batista de Sousa
Galileu.batista -at +ifrn -edu +br

Fundamentos de Listas

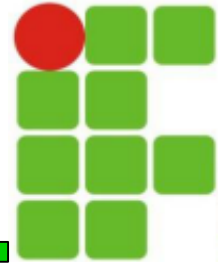


- Listas são **coleções** de dados
- Podem conter vários elementos
  - Cada elemento pode ser de um tipo diferente
  - Inclusive uma nova lista _

```
filhos   = ["Tiago", "Amanda", "Giovanna" ]
sobrinha = ["Bela", 10]
irmaos = [ ["Jr", 52], ["Riva", 43] ]
```

  - Muitos dos fundamentos de uso de *strings* se aplicam

Acesso a elementos de listas



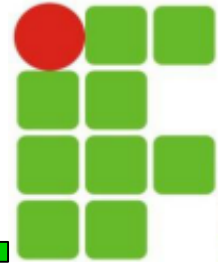
- É possível obter cada elemento de uma lista
  - Operação de indexação

[ ]

- Os índices iniciam em zero e vão até é o tamanho da lista – 1
  - Índices podem ser expressões
  - Índice fora de limites gera erro

```
pessoa = [“Amanda”, 27]
inicial = pessoa[0]
print (inicial)
```

Listas não são read-only



- É possível obter salvar um valor em uma posição da lista
  - Operação de indexação no lado esquerdo

[ ]

- Usa-se append para adicionar elementos em uma lista
  - O tipo do novo elemento pode ser qualquer.

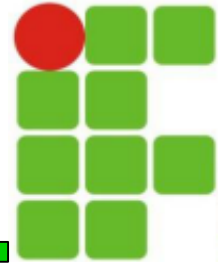
```
pessoa = [“Gio”, 16]
pessoa[1] = 17
print (pessoa)
```

```
[“Gio”, 17]
```

```
pessoa = []
pessoa.append (“Gio”)
pessoa.append (17)
print (pessoa)
```

```
[“Gio”, 17]
```

Operações básicas



- Concatenação (+)
  - Junta duas listas em uma
  - As originais permanecem

```
filha1 = ['Zi'] + [ 'Batista' ]
print (filha1)
```

- Replicação (*)
  - Cria uma lista como a repetição de outra

```
som = ["Toc"] * 3
print (som)
```

- Pertinência (in)
  - Elemento está na lista?

```
filhas = ["Gio", "Zi" ]
ehFilha = "Gio" in filhas
```

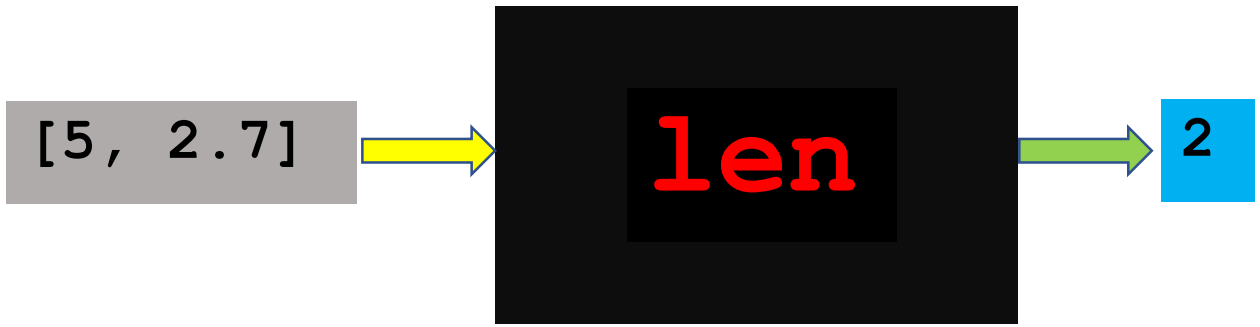
Tamanho de listas

- Há uma função predefinida que retorna o tamanho de uma lista

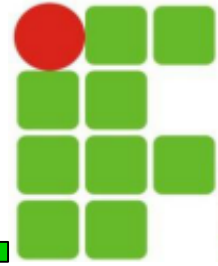
len(...)

- Uma função é um código que alguém já escreveu. Sorte sua.

```
zi = [ "Zi", 27 ]
print (len(zi))
```



Sublistas (*slicing*)



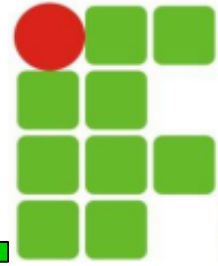
- A indexação retorna um elemento da lista
- O slicing retorna uma lista que é parte da original

[:]

- Informa-se o índice inicial e o final (não incluído)
  - Podem ser expressões

```
filho = ["Tiago", "Amanda", "Giovanna"]
elas  = nome[1:3]
print (elas)
```

Lembra de *range*?



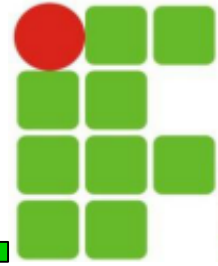
- Tecnicamente trata-se de um ***iterator***

- O *iterator* mais simples é **range**
  - Gera um conjunto de números inteiros
  - Especifica-se inicio, fim, incremento

Ideias similares para slicing

- Exemplos de *range*:
  - range (100) - gera números [0, 100 [ou *[0, 99]*
  - range (60, 100) - gera números [60, 100 [ou *[60, 99]*
  - range (3, 11, 2) - gera números (3, 5, 7, 9)

Navegação em listas com for

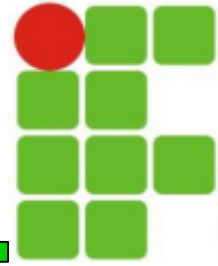


- Listas são coleções de dados, certo?
  - Uso de for é adequado e suportado

```
nomes = [“Tiago”, “Jane”, “Zi”, “Gio” ]
for nome in nomes:
    print (nome)
```

Em cada repetição do for nome recebe um novo elemento de nomes

Navegação em listas com for

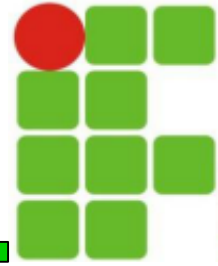


- Listas são coleções de dados, certo?
  - Uso de for é adequado e suportado

```
nomes = [“Tiago”, “Jane”, “Zi”, “Gio” ]
for pos in range(len(nomes)):
    print (nome[pos])
```

Em cada repetição do for pos recebe um índice de um elemento de nomes

Sublistas (*slicing*)



- É possível omitir o índice inicial – usa **zero**

```
idades = [ 51, 29, 52, 27, 17 ]
eles   = idades[:3]
print (eles)
```

→ `[ 51, 29, 52 ]`

- É possível omitir o índice final – usa o **len**

```
idades = [ 51, 29, 52, 27, 17 ]
elas   = idades[3:]
print (elas)
```

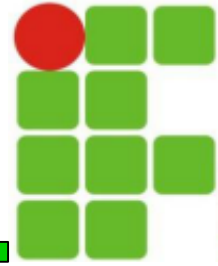
→ `[ 27, 17 ]`

- É possível informar um passo

```
idades = [ 51, 29, 52 ]
oque   = idades[:3:2]
print (oque)
```

→ `?`

Operações nativas



- Valor máximo (max)
  - Obtem o valor máximo em uma lista

```
idades = [29, 27, 17]
print ("O mais velho tem: ",
                max (idades))
```

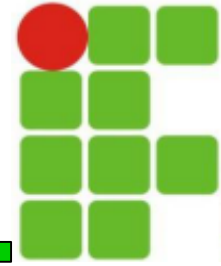
- Valor mínimo (min)
  - Obtem o valor mínimo em uma lista

```
idades = [29, 27, 17]
print ("O mais novo tem: ",
                min (idades))
```

- Soma (sum)
  - Soma elementos na lista

```
idades = [29, 27, 17]
print ("A soma das idades é:,
                sum (idades))
```

Strings vs Listas



- São irmãos gêmos, mas
  - Strings são *read-only*
  - Listas não são *read-only*

```
filha = "Zi Batista Sousa"
nomes = filha.split()
```

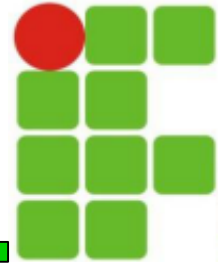
```
["Zi", "Batista", "Sousa" ]
```

- Possível quebrar um *string*
  - Gerando uma lista de strings
  - Operação das mais utilizadas
  - Possível especificar o separador, se omitido é o espaço

```
filha = "Zi Batista Sousa"
nomes = filha1.split('t')
```

```
[ "Zi Ba", "is", "a Sousa" ]
```

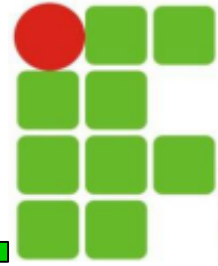
Outras funções/métodos?



```
>>> dir (list())
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']]
```

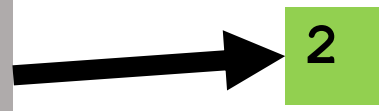
O que elas fazem: https://docs.python.org/3/tutorial/datastructures.html

Busca/contagem de ocorrências



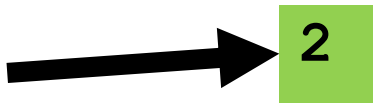
- O método ***index*** **encontra a primeira** ocorrência
  - Retorna o índice (-1, se não encontra)

```
nome = [ 3, 81, 13, 17 ]
pos_13 = nome.index(13)
print (pos_13)
```

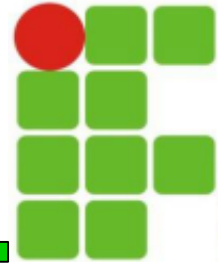


- O método count retorna o número de ocorrências

```
nome = [ 3, 11, 17, 13, 17 ]
oc17 = nome.count(17)
print (oc17)
```



Comprehensions



- Capacidade de criar listas a partir de coleções
- Muito sofistificado:

  [letra for letra in "Gio"]

```
["G", "i", "o"]
```

- Exemplos mais complexos:

```
idades = [52, 29, 27, 17]
idades202x = [idade+1 for idade in idades]
S = [ x**2 for x in range(20) ]
K = [ x//1024 for x in S if x % 1024 == 0]
```