VETORES (LISTAS E TUPLAS)

PROGRAMAÇÃO APLICADA A MATEMÁTICA

Meirylene Avelino
meirylenerea@id.uff.br

EXEMPLO MOTIVACIONAL

Programa para auxiliar a escrever “Parabéns!” nas melhores provas de uma disciplina com 3 alunos

- Ler os nomes e as notas de 3 alunos
- Calcular a média da turma
- Listar os alunos que tiveram nota acima da média

EXEMPLO MOTIVACIONAL

```
nome1 = input('Informe o nome do aluno 1: ')
nome2 = input('Informe o nome do aluno 2: ')
nome3 = input('Informe o nome do aluno 3: ')

nota1 = float(input('Informe a nota de ' + nome1 + ':'))
nota2 = float(input('Informe a nota de ' + nome2 + ':'))
nota3 = float(input('Informe a nota de ' + nome3 + ':'))

media = (nota1 + nota2 + nota3)/3
print('A media da turma foi', media)

if nota1 > media:
    print('Parabens', nome1)
if nota2 > media:
    print('Parabens', nome2)
if nota3 > media:
    print('Parabens', nome3)
```

E SE FOSSEM 40 ALUNOS?

- É possível definir variáveis que guardam mais de um valor de um mesmo tipo
- Essas variáveis são conhecidas como variáveis compostas, variáveis subscritas, variáveis indexáveis ou arranjos (array)
- Em Python existem três tipos principais de variáveis compostas:
  - Listas
  - Tuplas
  - Dicionários

VETORES

- Variável composta **unidimensional**
  - Contém espaço para armazenar diversos valores
  - É acessada via um índice

- A ideia de vetor é comum na matemática, com o nome de variável subscrita
  - Exemplo: x1 , x2 , ..., xn

VETORES

- O que vimos até agora são variáveis com somente um valor
  - Exemplo: y = 123

- No caso de vetores, uma mesma variável guarda ao mesmo tempo múltiplos valores

Exemplo: $x_1 = 123, x_2 = 456, \ldots$

$x = [123, 456, \ldots]$

LISTAS

- Em outras linguagens de programação, listas são chamadas de **vetores** e possuem restrições que Python não impõe:

- Em Python, os valores de uma lista podem ser de qualquer tipo
- Em outras linguagens, os valores precisam ser do mesmo tipo

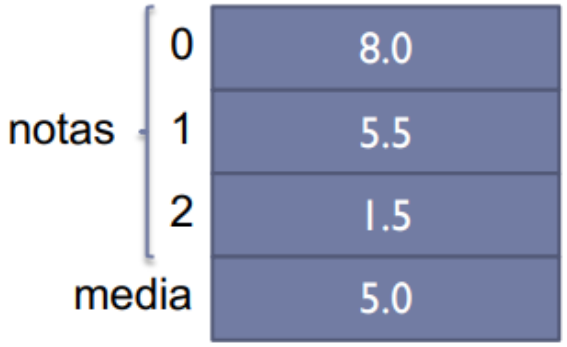
Em Python

- lista = ['A', 1, 2, 'Casa', 2.3]
- notas = [10, 5, 6.7, 2, 7.5]

UTILIZAÇÃO DE LISTAS

- Para acessar (ler ou escrever) uma posição do vetor, basta informar a posição entre colchetes

```
notas = [8.0, 5.5, 1.5]
media = (notas[0] + notas[1] + notas[2]) / 3
```



UTILIZAÇÃO DE LISTAS

- Pode-se iterar por todos os seus valores usando um comando for

```
notas = [8.0, 5.5, 1.5]
for i in range(3):
    print(notas[i])
```

CRIAÇÃO DE UMA LISTA A PARTIR DE VALORES LIDOS DO TECLADO

- Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado.

```
notas[0] = float(input('Digite a nota do primeiro aluno: '))
notas[1] = float(input('Digite a nota do segundo aluno: '))
notas[2] = float(input('Digite a nota do terceiro aluno: '))
```

CRIAÇÃO DE UMA LISTA A PARTIR DE VALORES LIDOS DO TECLADO

- Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado.

```
notas[0] = float(input('Digite a nota do primeiro aluno: '))
notas[1] = float(input('Digite a nota do segundo aluno: '))
notas[2] = float(input('Digite a nota do terceiro aluno: '))
```

```
Digite a nota do primeiro aluno: 8
Traceback (most recent call last):
  File "/Users/vanessa/workspace/PyCharmProjects/AloMundo/notas.py",
line 1, in <module>
    notas[0] = float(input('Digite a nota do primeiro aluno: '))
NameError: name 'notas' is not defined

Process finished with exit code 1
```

É PRECISO PRIMEIRO CRIAR A LISTA...

- Como não sabemos o que colocar em cada posição da lista, vamos criar uma lista vazia

```
notas = []
```

- Depois vamos adicionar valores na lista usando **append**

```
n = float(input('Digite a nota do primeiro aluno: '))
notas.append(n)
```

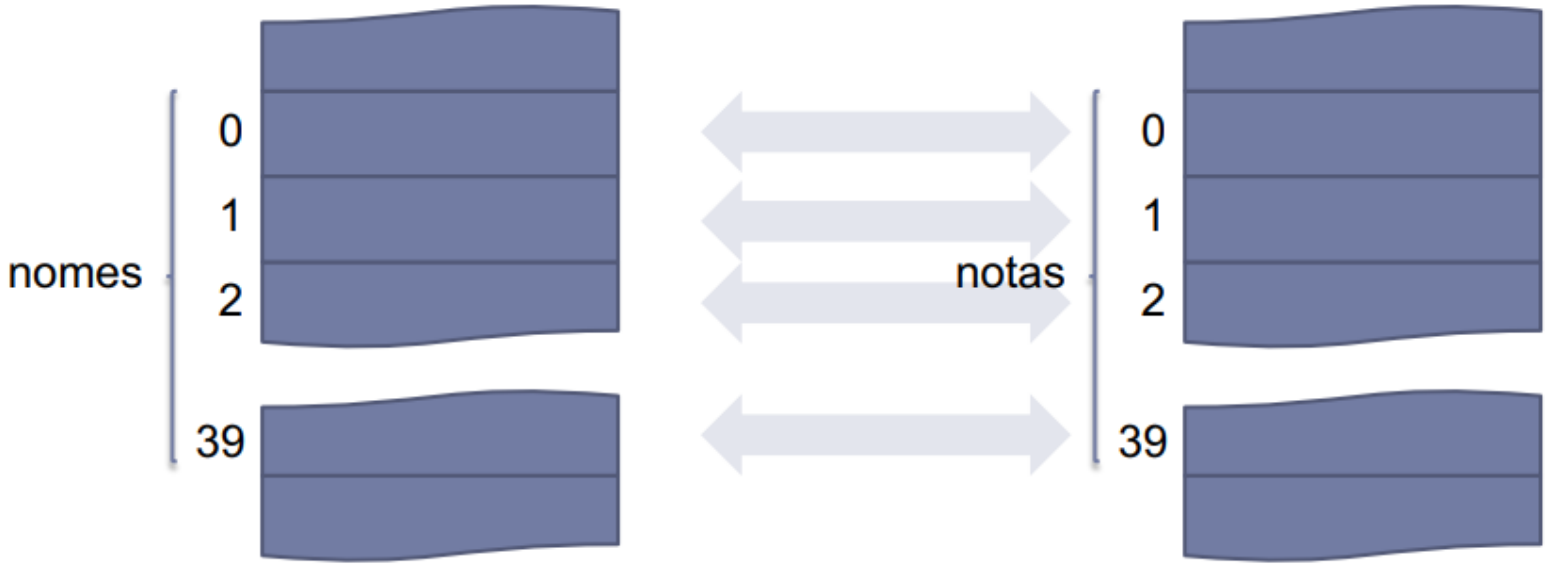
VOLTANDO AO EXEMPLO

- Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado.

```
notas = []
notas.append(float(input('Digite a nota do primeiro aluno: ')))
notas.append(float(input('Digite a nota do segundo aluno: ')))
notas.append(float(input('Digite a nota do terceiro aluno: ')))
print(notas)
```

RETOMANDO: E SE FOSSEM 40 ALUNOS?

- Criaríamos dois vetores (nomes e notas) de 40 posições
- Vincularíamos a posição i do vetor de nomes à posição i do vetor de notas



RETOMANDO: E SE FOSSEM 40 ALUNOS?

```
num_alunos = 40
nomes = []
notas = []
media = 0

for i in range(num_alunos):
    nomes.append(input('Informe o nome do aluno: '))
    notas.append(float(input('Informe a nota de ' + nomes[i] + ': ')))
    media = media + notas[i]

media = media / num_alunos
print('A media da turma eh ', media)

for i in range(num_alunos):
    if notas[i] > media:
        print('Parabens', nomes[i])
```

CUIDADOS NO USO DE LISTAS

- Certifique-se de que não esteja querendo acessar posição da lista que não existe

- Exemplo:

```
alunos = ['Andre', 'Lucas', 'Antonio', 'Maria']
print(alunos[4])
```

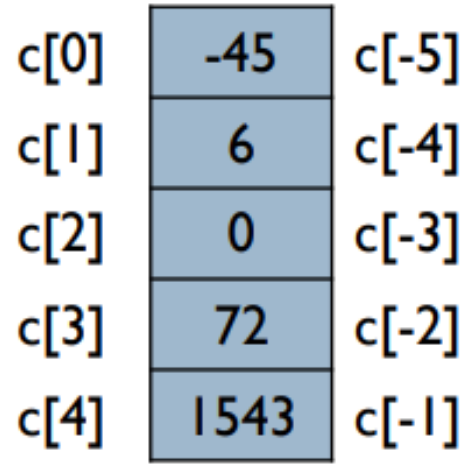
```
Traceback (most recent call last):
  File "/Users/vanessa/workspace/PyCharmProjects/AloMundo/notas.py",
line 2, in <module>
    print(alunos[4])
IndexError: list index out of range

Process finished with exit code 1
```

ÍNDICES PARA ACESSO AOS ELEMENTOS DA LISTA

- Python permite acesso à lista em ordem crescente ou decrescente de posição
  - Primeira posição é 0
  - Última posição é -1

```
>>> c = [-45, 6, 0, 72, 1543]
>>> c[3]
72
>>> c[-2]
72
>>> c[0] = c[-5]
True
```



FUNÇÕES DE MANIPULAÇÃO DE LISTAS

- len(lista)
  - Retorna o tamanho da lista

```
>>> numeros = [3,1,6,7,10,22,4]
>>> len(numeros)
7
```

EXEMPLO

- Programa que lê uma lista do teclado, soma 1 aos elementos da lista e imprime a lista resultante

```
continua = True
lista = []
while (continua):
    n = int(input('Digite um numero: '))
    lista.append(n)
    op = input('Deseja continuar? (s/n): ')
    if op !== 's' and op != 'S':
        continua = False

print(lista)

for i in range(len(lista)):
    lista[i] = lista[i] + 1

print(lista)
```

CONCATENAÇÃO DE LISTAS

- É possível anexar os valores de uma lista em outra usando o operador "+"

```
>>> lista = [1,2,3]
>>> lista = lista + [4]
[1,2,3,4]
>>> lista = lista + [4,5,6]
[1,2,3,4,4,5,6]
```

EXEMPLO

- Programa que retorna uma lista com todos os números pares entre 2 e um número n, inclusive

```
n = int(input('Digite um numero: '))
lista = []
for i in range(2,n+1,2):
    lista = lista + [i]
print(lista)
```

EXEMPLO

- Programa que retorna uma lista com todos os números pares entre 2 e um número n, inclusive, em ordem reversa

```
n = int(input('Digite um numero: '))
lista = []
for i in range(2,n+1,2):
    lista = [i] + lista
print(lista)
```

"MULTIPLICAÇÃO" DE LISTAS

- O operador "*" repete **n** vezes os elementos que já estão na lista
- lista * n equivale a lista + lista + ... + lista (n vezes)

```
>>> lista = [1,2,3]
>>> lista = lista * 3
[1,2,3,1,2,3,1,2,3]
```

INICIALIZAÇÃO DE LISTAS COM ZERO

- Em diversas situações onde já sabemos de antemão qual será o tamanho de uma lista de inteiros, é útil inicializar a lista com o valor 0
- Isso evita que precisemos usar o append para adicionar valores

```
>>> tamanho = 10
>>> lista = [0] * tamanho
>>> lista
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

EXEMPLO

```
# inicializa vetor de notas com 0
notas = [0] * 3
soma = 0
# preenche vetor de notas, sem usar append
for i in range(3):
    notas[i] = float(input("Digite a nota do aluno " +
str(i) + ": "))
    soma = soma + notas[i]
print("A media da turma é", soma/3)
```

TESTE DE PERTINÊNCIA

- Retornar True caso o valor 10 pertença à lista, e False caso contrário

```
lista = [1, 2, 3, 4]
valor = 7
resultado = False
for i in range(len(lista)):
    if lista[i] == valor:
        resultado = True
print(resultado)
```

```
False
```

ALTERNATIVA: ELEMENTO IN LISTA

```
lista = [1, 2, 3, 4]
resultado = 7 in lista
print(resultado)
```

```
False
```

CONHECIMENTO ÚTIL: SPLIT RETORNA UM VETOR

```
>>>x = input("Digite valores
separados por espaços: ").split()
Digite valores separados por espaços:
10 20 30 40
>>>x
['10', '20', '30', '40']
>>>x[0]
'10'
```

REPRESENTAÇÃO DE LISTAS EM MEMÓRIA

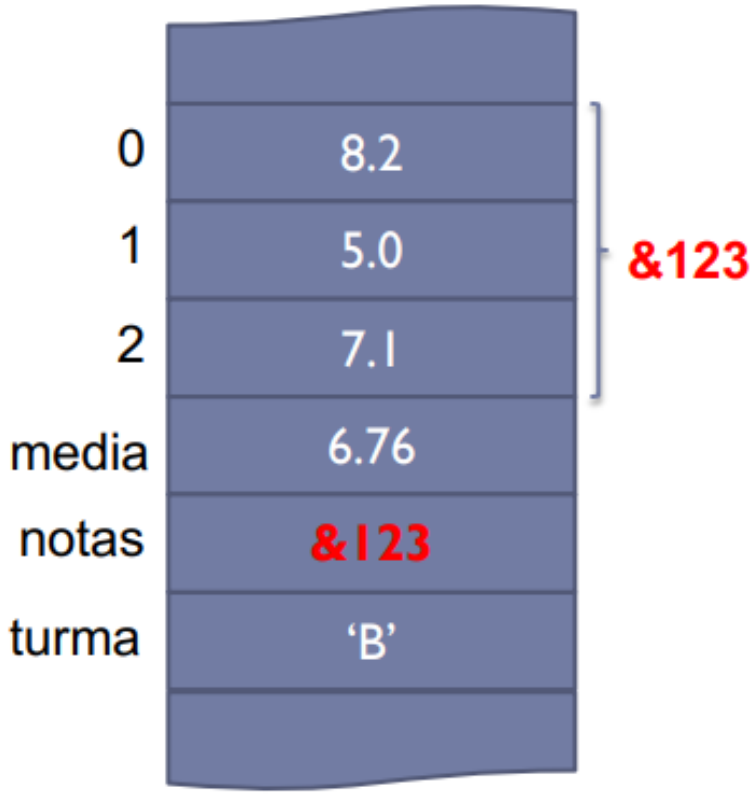
- O valor de uma variável de lista na verdade é um endereço de memória

REPRESENTAÇÃO DE LISTAS EM MEMÓRIA

Em Phyton

```
notas = [8.2, 5.0, 7.1]
turma = 'B'
media = 0
for i in range(len(notas)):
    media = media + notas[i]
media = media/len(notas)
```

Na Memória



CÓPIA DE LISTAS

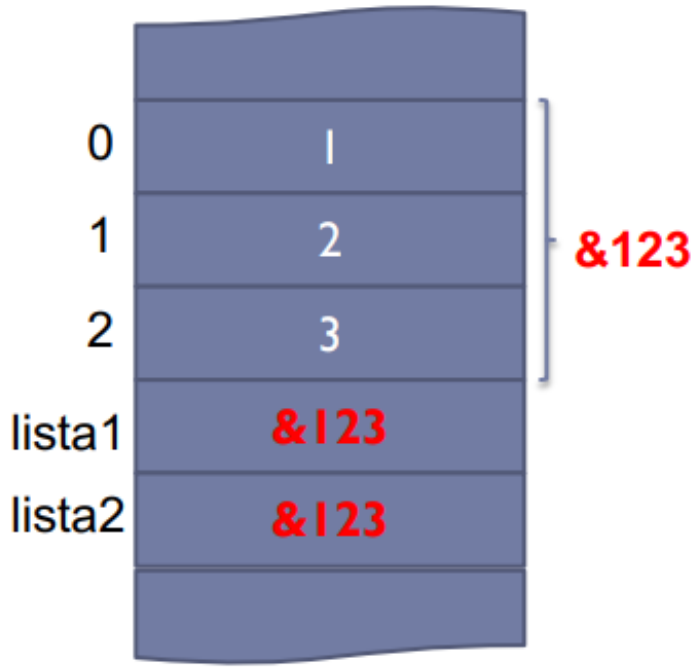
- Ao copiar uma lista para outra, o que é feito é copiar o valor do endereço de memória
  - Ambas passam a apontar para o mesmo endereço, portanto o que for modificado em uma lista também será modificado na outra

CÓPIA DE LISTAS

Em Phyton

```
>>> lista1 = [1, 2, 3]
>>> lista2 = lista1
```

Na Memória

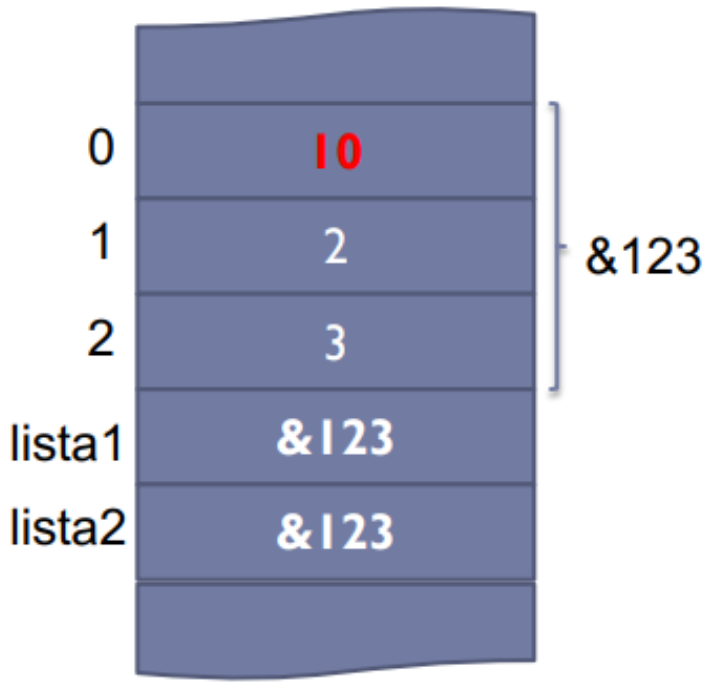


CÓPIA DE LISTAS

Em Phyton

```
>>> lista1 = [1, 2, 3]
>>> lista2 = lista1
>>> lista1[0] = 10
>>> lista1
[10, 2, 3]
>>> lista2
[10, 2, 3]
```

Na Memória

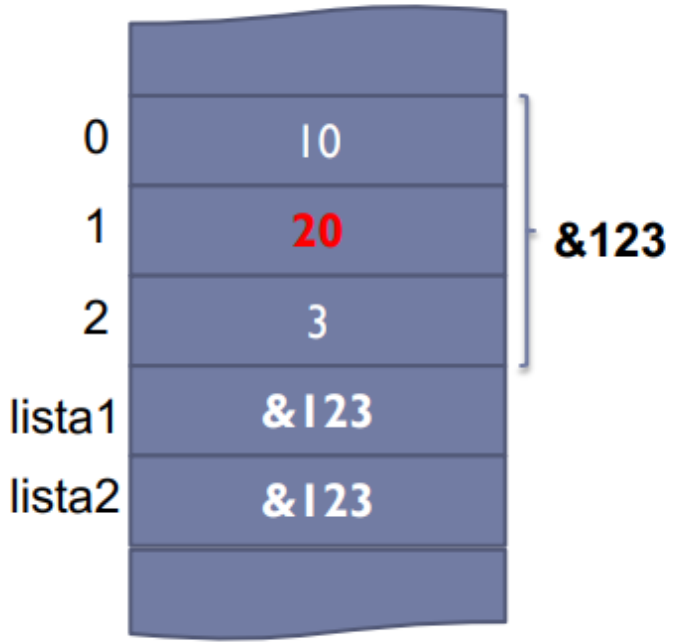


CÓPIA DE LISTAS

Em Phyton

```
>>> lista1 = [1, 2, 3]
>>> lista2 = lista1
>>> lista1[0] = 10
>>> lista1
[10, 2, 3]
>>> lista2
[10, 2, 3]
>>> lista2[1] = 20
>>> lista2
[10, 20, 3]
>>> lista1
[10, 20, 3]
```

Na Memória

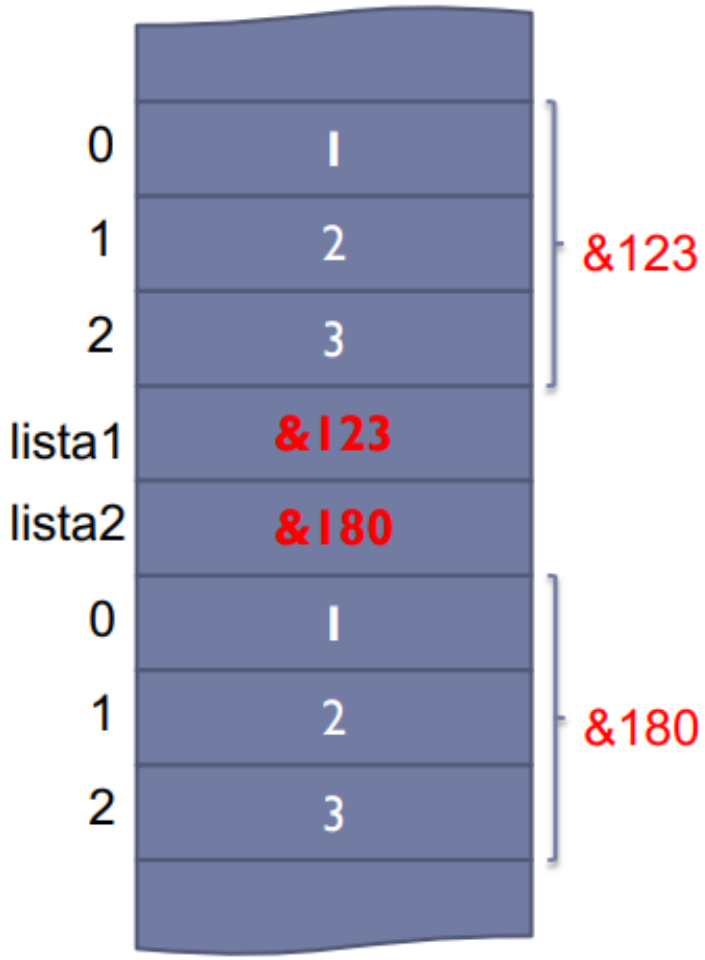


COMO EVITAR ISSO?

- Usar um for para copiar valor a valor

EXEMPLO

```
>>> lista1 = [1, 2, 3]
>>> lista2 = []
>>> for i in range(len(lista1)):
...     lista2.append(lista1[i])
...
```



DESSA FORMA

- Alterações em uma lista não são refletidas na outra

```
>>> lista1 = [1, 2, 3, 4, 5]
>>> for i in range(len(lista1)):
...     lista2.append(lista1[i])
>>> lista2[0] = 10
>>> lista1
[1, 2, 3, 4, 5]
>>> lista2
[10, 2, 3, 4, 5]
>>> lista1[3] = 20
>>> lista2
[10, 2, 3, 4, 5]
```

TUPLAS

- Tuplas são sequências de valores, da mesma forma que listas

- Mas, existem diferenças...
  - Os valores de uma tupla, ao contrário de uma lista, são imutáveis
  - Tuplas usam parênteses enquanto listas usam colchetes

```
>>> lista = [1, 2, 3, 4]
>>> tupla = (1, 2, 3, 4)
```

TUPLAS

- Tupla vazia

```
>>> tupla = ()
```

- Tupla com um único elemento (note a necessidade da vírgula, mesmo sendo um único elemento)

```
>>> tupla = (1,)
```

ACESSO AOS ELEMENTOS DE UMA TUPLA

- Acesso é feito pela posição, da mesma forma que nas listas

```
>>> tupla = ("Maria", "Joao", "Carlos")
>>> tupla[0]
"Maria"
```

ATUALIZAÇÃO DE TUPLAS

- Como são imutáveis, não é permitido atualizar os valores dentro de uma tupla

```
>>> tupla = ("Maria", "Joao", "Carlos")
>>> tupla[0] = "Ana"
TypeError: 'tuple' object does not support
item assignment
```

OPERADORES BÁSICOS SOBRE TUPLAS

Expressão	Resultado	Descrição
len((1,2,3))	3	Número de elementos que a tupla contém
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenação
(1,) * 4	(1,1,1,1)	Repetição
3 in (1, 2, 3)	True	Pertencimento
for x in (1,2,3): print(x)	1 2 3	Iteração

REFERÊNCIAS

- Slides baseados no curso da Vanessa Murta

VETORES (LISTAS E TUPLAS)

PROGRAMAÇÃO APLICADA A MATEMÁTICA

Meirylene Avelino
meirylenerea@id.uff.br