Universidade de Brasília
Departamento de Ciência da Computação

Semana 12

Listas (parte 2)

CIC0004
Algoritmos e Programação de Computadores

Prof. Pedro Garcia Freitas https://pedrogarcia.gitlab.io/ pedro.garcia@unb.br

Brasilia

Objetivos

Esta aula continua o conceito de coleções (listas e tuplas) e demonstrar como esse conceito é usável em linguagem Python.

21. Extração de Elementos de Listas

- Extrair elementos um-a-um
- `lista.pop(i)`
  - Remove o elemento `lista[i]`
  - Retorna o **elemento removido**!
- `lista.pop()`
  - Remove o **ultimo elemento** da lista!

21. Extração de Elementos de Listas

- Exemplo

```
>>> mylist = [11, 22, 33, 44, 55, 66]
>>> mylist.pop(0)
11
>>> mylist
[22, 33, 44, 55, 66]
```

21. Extração de Elementos de Listas

- Exemplo

```
>>> mylist.pop()
66
>>> mylist
[22, 33, 44, 55]
>>> mylist.pop(2)
44
>>> mylist
[22, 33, 55]
```

21. Extração de Elementos de Listas

- Exemplo 2

```
>>> waitlist = ['Ann', 'Cristi', 'Dean']
>>> waitlist.pop(0)
'Ann'
>>> waitlist
['Cristi', 'Dean']
>>> waitlist.append('David')
>>> waitlist
['Cristi', 'Dean', 'David']
```

22. Reordenando uma lista

- `lista.sort()`
  - Ordena a lista (*inplace*)
  - Não retorna nada!

22. Reordenando uma lista

- Exemplo:

```
>>> names = ['Alice', 'Carol', 'Bob']
>>> names.sort()
>>> print(names)
['Alice', 'Bob', 'Carol']
```

- **names.sort()** é um ***método*** *da lista* **names**
  - ***In-place sort***

22. Reordenando uma lista

- Exemplo 2:

```
>>> mylist = [11, 12, 13, 14, 'done']
>>> mylist.sort()
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
 mylist.sort()
TypeError: unorderable types: str() < int()
```

  - *Não podemos comparar tipos diferentes*

22. Reordenando uma lista

- Exemplo 3:

```
>>> newlist = [0, -1, +1, -2, +2]
>>> newlist.sort()
>>> print(newlist)
[-2, -1, 0, 1, 2]
```

  - ***In-place sort***

22. Reordenando uma lista

- Exemplo 4 (sorted):

```
>>> newlist = [0, -1, +1, -2, +2]
>>> sortedlist= sorted(newlist)
>>> print(newlist)
[0, -1, 1, -2, 2]
>>> print(sortedlist)
[-2, -1, 0, 1, 2]
```

  - **sorted(…)** é uma **retorna uma lista**

23. Revertendo uma lista

- `lista.reverse()`
  - Reverte a ordem atual dos elementos
  - Não retorna nada!

23. Revertendo uma lista

- Exemplo:

```
>>> names = ['Alice', 'Bob', 'Carol']
>>> names.reverse()
>>> names
['Carol', 'Bob', 'Alice']
```

- **names.reverse()** é um **método** da lista **names**

24. Pesquisa e recuperação

- ***mylist*.index(*item*)**
  - Retorna a posição da **primeira ocorrência** do item na lista
- ***mylist*.count(*item*)**
  - Retorna o **número de ocorrências** do item na lista
- ***mylist*.remove(*item*)**
  - ***Remove*** a primeira ocorrência do item na lista!

24. Pesquisa e recuperação

- Exemplo:

```
>>> names
['Ann', 'Carol', 'Bob', 'Alice', 'Ann']
>>> names.index('Carol')
1
>>> names.count('Ann')
2
>>> names.remove('Ann')
>>> names
['Carol', 'Bob', 'Alice', 'Ann']
```

25. Função `sum`

- Exemplo:

```
>>> list = [10, 20 ,20]
>>> sum(list)
  50
>>> list = ['Alice', ' and ', 'Bob']
>>> sum(list)
```

  - **Não** funciona!

25. Função `sum`

- Exemplo 2: média

```
>>> prices = [1.899, 1.959, 2.029, 2.079]
>>> sum(prices)
7.966000000000001
>>> len(prices)
4
>>> sum(prices)/len(prices) # Média
1.9915000000000003
```

26. Appending vs. Concatenating

- Append: ***nova entrada (valor)*** na lista (mutavel)

```
>>> names = ['Ann', 'Bob', 'Alice']
>>> names.append('Carol')
>>> names
['Ann', 'Bob', 'Alice', 'Carol']
```

- A lista **`names`** foi atualizada!

26. Appending vs. Concatenating

- + (concatenate): Operação gera uma nova lista (imutável)

```
>>> names = ['Ann', 'Bob', 'Alice']
>>> names + ['Carol']
['Ann', 'Bob', 'Alice', 'Carol']
>>> names
['Ann', 'Bob', 'Alice'] # Mesmo estado
>>> names = names +['Carol']
>>> names
['Ann', 'Bob', 'Alice', 'Carol'] # OK
```

27. Listas e *for loops*

- ***Item a item***

  - `for item in lista:`
    - Processa sucessivamente cada item na lista.

27. Listas e *for loops*

- ***Iteração baseada na posição***

```
for index in range(len(lista)) :
```

  - Passa pelos índices de todas as entradas da lista
  - Nos dá acesso às suas posições

27. Listas e *for loops*

- Exemplo:

```
>>> names
['Ann', 'Bob', 'Alice', 'Carol']
>>> for item in names:
    print(item)
Ann
Bob
Alice
Carol
```

27. Listas e *for loops*

- Exemplo 2:

```
>>> names
['Ann', 'Bob', 'Alice', 'Carol']
>>> for i in range(len(names)):
  print(names[i])
Ann
Bob
Alice
Carol
```

27. Listas e *for loops*

- Exemplo 3:

```
>>> scores = [50, 80, 0, 90]
>>> def count_zeroes(lista):
        count = 0
        for item in lista:
            if item == 0 :
                count += 1
        return count
>>> count_zeroes(scores)
1
```

27. Listas e *for loops*

- Exemplo 4:

```
>>> scores = [50, 80, 0, 90]
>>> def another_count_zeroes(lista):
        count = 0
        for index in range(len(lista)) :
            if lista[index] == 0 :
                count += 1
        return count
>>> another_count_zeroes(scores)
1
```

27. Listas e *for loops*

- Exemplo 5:

```
>>> scores = [50, 80, 0, 90]
>>> def yet_another_count_zeroes(lista):
        return sum(1 for i in lista if i == 0)
>>> yet_another_count_zeroes(scores)
1
```

28. Listas como argumentos de função

- Podemos modificar uma lista dentro de uma função (passagem por referências)

```
def capitalizeList(l) :
    for i in range(len(l)) :
        l[i] = l[i].capitalize()
names = ['Ann', 'bob', 'lIZ']
capitalizeList(names)
print(names)
```

```
['Ann', Bob', 'Liz']
```

28. Listas como argumentos de função

- Devemos cuidar se queremos evitar side effects

```
def capitalize_list(l) :
    newlist = []
    for item in l :
        newlist.append(item.capitalize())
    return newlist
names = ['ann', 'Bob', 'lIZ']
print(capitalize_list(names))
```

```
['Ann', Bob', 'Liz']
```

28. Listas como argumentos de função

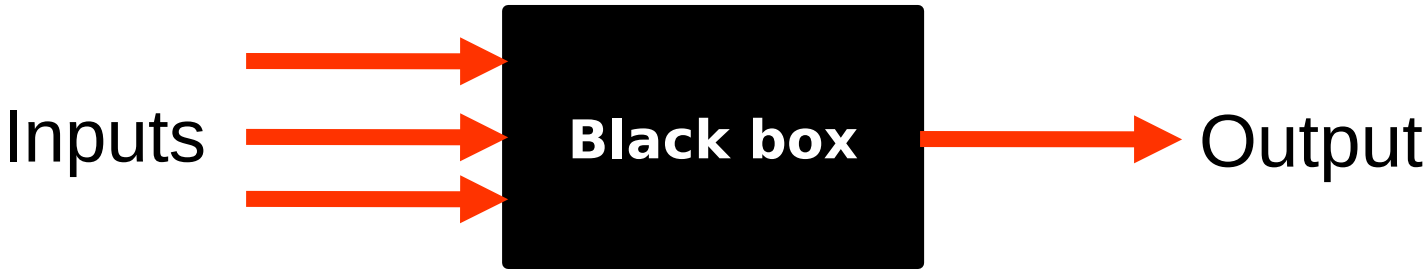
```
def capitalize_list_2(l) :
    return [i.capitalize() for i in l]

names = ['ann', 'Bob', 'liz']
print(capitalize_list_2(names))
```

```
['Ann', Bob', 'Liz']
```

29. Funções Puras

- Sempre retornam um valor
- Não produzem *side-effects*
  - São como funções matemáricas
- Implementam uma “caixa preta”



29. Funções Puras

- Vantagens:
  - Mais fáceis de se entender
  - Preferível sempre que possível

29. Funções Puras

- Exemplo: outra função retornando uma lista

```
def militime_to_hours_and_minutes(miltime):
    hours = miltime // 100
    minutes = miltime % 100
    return [hours, minutes]

[hh, ss] = militime_to_hours_and_minutes(1538)
print(hh)
print(ss)
```

29. Funções Puras

- Exemplo: Retorna a lista de primos até *"n"*

```
def primes_upto(n) :
    result = []
    for i in range(2, n):
        if is_prime(i) :
            result.append(i)
    return result
```

29. Fu

- Exe

```
def is_prime(n):
    if (n <= 1):
        return False
    else:
        for i in range(2, int(math.sqrt(n))+1):
            if (n % i == 0):
                return False
    return True
```

```
def
    result
    for i in     ge(2, n):
        if is_prime(i) :
            result.append(i)
    return result
```

Exercício

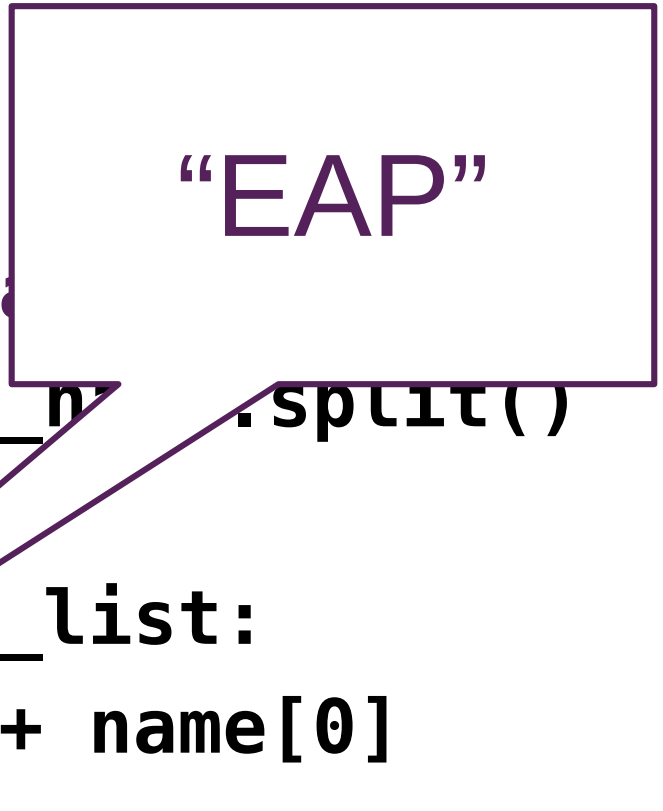
- Qual o resultado?

```
full_name = "Edgar Allan Poe"
name_list = full_name.split()
init = ""
for name in name_list:
    init = init + name[0]
print(init)
```

Exercício

- Qual o resultado?

```
full_name = "Edg
name_list = full_n    .split()
init = ""
for name in name_list:
    init = init + name[0]
print(init)
```



30. Lists comprehensions

- Inicialização de listas:
- ```
  >>> [ 0 for n in range(0, 9)]
  [0, 0, 0, 0, 0, 0, 0, 0, 0]
  ```
- ```
  >>> [n for n in range(1,11)]
  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  ```
- ```
  >>> [2*n+1 for n in range(0, 6)]
  [1, 3, 5, 7, 9, 11]
  ```
  ```

30. Lists comprehensions

- A cláusula `for n` ***é essencial!***

- **`[0 in range(0, 10)]`**
  `[True]`
  - Because **`0 is in range(0, 10)`**

30. Lists comprehensions

- Exemplos:

```
>>> [ c for c in 'Cougars']
[ 'C', 'o', 'u', 'g', 'a', 'r', 's']

>>> names = ['Ann', 'bob', 'liz']

>>> new =  [n.capitalize() for n in names]

>>> new
['Alice', 'Bob', 'Liz']

>>> names
['Alice', 'bob', 'liz']
```

30. Lists comprehensions

- Exemplos:

```
>>> first5 = [n for n in range(1,6)]
>>> first5
[1, 2, 3, 4, 5]
>>> first5squares = [n*n for n in first5]
[1, 4, 9, 16, 25]
```

30. Lists comprehensions

- Uma equivalência:

```
[n*n for n in range(1, 6)]
[1, 4, 9, 16, 25]
```

equivale a

```
a = []
for n in range(1,6) :
    a.append(n*n)
```

30. Lists comprehensions

- Filtered comprehensions

```
>>> a = [11, 22, 33, 44, 55]
>>> b = [ n for n in a if n%2 == 0]
>>> b
[22, 44]
>>> c = [n//11 for n in a if n > 20]
>>> c
[2, 3, 4, 5]
```

30. Lists comprehensions

- Mais exemplos de *comprehensions*

```
>>> s =
[['Ann', 'CS'],['Bob', 'CE'],['Liz', 'CS']]
>>> [x[0] for x in s if x[1] =='CS']
['Ann', 'Liz']
>>> sum([1 for _ in s if _[1] =='CS'])
2
```

31. Listas aninhadas (listas de listas)

```
>>> nested = [['Ann', 90], ['Bob', 84]]
>>> innerlist = nested[1]
>>> innerlist
['Bob', 84]
>>> innerlist[0]
'Bob'
>>> nested[1][0]
'Bob'
```

32. Strings e listas

```
>>> s = 'No pain, no gain'
>>> lst1 = s.split()
>>> lst1
['No', 'pain,', 'no', 'gain']
>>> lst2 = s.split(',')
>>> lst3 = s.split('ai')
>>> lst3
['No p', 'n no g', 'n']
```

32. Strings e listas

- A função “list” se aplica a strings

```
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKURATU
YAQHANYELAY
SUKSAMA
EKHMET
BİYAN
SHUKRIA
TINGKI
GRAZIE
MEHRBANI
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
PALDIES
BOLZÏN
MERCI

Dúvidas?