Revisitando Listas

MC102-2018s1-Aula11-180410

Arthur J. Catto, PhD
ajcatto@g.unicamp.br

10 de abril de 2018

1 Revisitando Listas

Listas, dicionários, tuplas e conjuntos são as quatro estruturas de dados básicas implementadas por Python.

Uma rápida introdução a listas foi vista na *Aula 07*. Aqui vamos estender um pouco mais esse conhecimento.

Uma ***lista*** é uma ***sequência de objetos***, ***ordenada***, ***mutável***, ***iterável*** e ***não necessariamente homogênea***.

Cada elemento de uma lista é identificado por um *índice* que indica sua posição na sequência.

1.1 Operações com listas

1.1.1 Criar uma lista

Uma lista vazia pode ser criada por uma atribuição simples:

```
In [44]: nums = []
         'nums', id(nums), nums

Out[44]: ('nums', 4493296328, [])
```

Todo objeto em Python tem uma identificação única. A chamada de função `id(nums)` que aparece no `print` acima exibe o `identificador` do objeto que está associado a `nums` no momento da chamada. Nós vamos usar essa informação várias vezes durante esta aula.

O operador `+` concatena duas listas enquanto o operador `*` replica uma lista um certo número de vezes.

```
In [45]: n = 1
         nums = [0] + 3 * [n] + n * [2]
         'nums', id(nums), nums

Out[45]: ('nums', 4495044104, [0, 1, 1, 1, 2])
```

A função `len` retorna o número de elementos numa lista.

```
In [46]: len(nums)

Out[46]: 5
```

1.1.2 Alterar um ou mais elementos de uma lista

É possível alterar os valores associados a quaisquer elementos de uma lista. Por exemplo, ...

```
In [47]: nums = [1, 3, 4, 7, 9]
         'nums', id(nums), nums

Out[47]: ('nums', 4495554312, [1, 3, 4, 7, 9])

In [48]: nums[2] = 5
         'nums', id(nums), nums

Out[48]: ('nums', 4495554312, [1, 3, 5, 7, 9])

In [49]: for i in range(5):
             nums[i] *= 2
         'nums', id(nums), nums

Out[49]: ('nums', 4495554312, [2, 6, 10, 14, 18])
```

1.1.3 Inserir um item no fim de uma lista

Vamos criar uma lista de números...

```
In [50]: nums = [1, 3, 4, 7]
         'nums', id(nums), nums

Out[50]: ('nums', 4495554888, [1, 3, 4, 7])
```

Há mais de uma maneira de inserir um item no fim dessa lista...

```
In [51]: nums += [9]
         'nums', id(nums), nums

Out[51]: ('nums', 4495554888, [1, 3, 4, 7, 9])

In [52]: nums.append(4)
         'nums', id(nums), nums

Out[52]: ('nums', 4495554888, [1, 3, 4, 7, 9, 4])
```

Note que o identificador associado à lista `nums` não se altera quando acrescentamos novos itens à lista.

1.1.4 Inserir um item numa posição qualquer

Também é possível inserir um novo item numa posição qualquer, inclusive no início e no fim de uma lista.

```
In [53]: nums = [1, 3, 4, 7]
         print('nums', id(nums), nums)

nums 4495689928 [1, 3, 4, 7]
```

```
In [54]: nums.insert(2, 99)
         print('nums', id(nums), nums)

nums 4495689928 [1, 3, 99, 4, 7]


In [55]: nums.insert(0, 99)
         print('nums', id(nums), nums)

nums 4495689928 [99, 1, 3, 99, 4, 7]


In [56]: nums.insert(len(nums), 99)
         print('nums', id(nums), nums)

nums 4495689928 [99, 1, 3, 99, 4, 7, 99]
```

1.1.5 Estender uma lista acrescentando ao final todos os itens de uma outra lista

```
In [57]: nums1 = [1, 3, 7]
         nums2 = [7, 8]
         print('nums1', id(nums1), nums1)
         print('nums2', id(nums2), nums2)

nums1 4495555400 [1, 3, 7]
nums2 4495553160 [7, 8]


In [58]: nums1.extend(nums2)
         print('nums1', id(nums1), nums1)
         print('nums2', id(nums2), nums2)

nums1 4495555400 [1, 3, 7, 7, 8]
nums2 4495553160 [7, 8]


In [59]: nums1 += nums2
         print('nums1', id(nums1), nums1)
         print('nums2', id(nums2), nums2)

nums1 4495555400 [1, 3, 7, 7, 8, 7, 8]
nums2 4495553160 [7, 8]
```

Nestes casos, note que os identificadores das duas listas continuam os mesmos e que a lista num2 não se altera.

O argumento de `extend` também pode ser uma constante...

```
In [60]: nums1.extend([0, 1])
         print('nums1', id(nums1), nums1)

nums1 4495555400 [1, 3, 7, 7, 8, 7, 8, 0, 1]
```

```
In [61]: nums2[len(nums2):] = [99]
         print('nums2', id(nums2), nums2)

nums2 4495553160 [7, 8, 99]


In [62]: nums2[len(nums2):] = [1, 2]
         print('nums2', id(nums2), nums2)

nums2 4495553160 [7, 8, 99, 1, 2]
```

1.1.6 Remover de uma lista o primeiro item com um dado valor

```
In [63]: nums = [1, 3, 4, 3, 7]
         nums.remove(3)
         print('nums', id(nums), nums)

nums 4495682312 [1, 4, 3, 7]
```

1.1.7 Remover e retornar o item numa dada posição

```
In [64]: x = nums.pop(2)
         print('x', x)
         print('nums', id(nums), nums)

x 3
nums 4495682312 [1, 4, 7]
```

Se o argumento for omitido, pop remove e retorna o último item da lista.

```
In [65]: x = nums.pop()
         print('x', x)
         print('nums', id(nums), nums)

x 7
nums 4495682312 [1, 4]
```

1.1.8 Remover todos os itens de uma lista

```
In [66]: nums = [1, 3, 4, 3, 7]
         print('nums', id(nums), nums)

nums 4495688904 [1, 3, 4, 3, 7]


In [67]: nums.clear()
         print('nums', id(nums), nums)

nums 4495688904 []
```

1.1.9 Obter o índice do primeiro item com um dado valor

```
In [68]: nums = [1, 3, 4, 3, 7]
         ix = nums.index(3)
         print('index(3)', ix)
         print('nums', id(nums), nums)
```

```
index(3) 1
nums 4495731976 [1, 3, 4, 3, 7]
```

```
In [69]: ix = nums.index(99)  # vai dar erro...
         print('index(99)', ix)
         print('nums', id(nums), nums)
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-69-c39c2dc448d7> in <module>()
    ----> 1 ix = nums.index(99)  # vai dar erro...
          2 print('index(99)', ix)
          3 print('nums', id(nums), nums)


        ValueError: 99 is not in list
```

1.1.10 Retornar o número de vezes que um dado valor aparece numa lista

```
In [70]: nums = [1, 3, 4, 3, 7]
         print('nums', id(nums), nums)

         c3 = nums.count(3)
         print('count(3)', c3)

         c9 = nums.count(9)
         print('count(9)', c9)
```

```
nums 4495688392 [1, 3, 4, 3, 7]
count(3) 2
count(9) 0
```

1.1.11 Fazer uma busca linear numa lista

O motivo de uma busca geralmente é encontrar um item da lista que satisfaça uma determinada condição. A busca linear começa em uma das extremidades da lista e caminha na direção da outra extremidade até encontrar o item desejado ou esgotar a lista. À medida que a busca avança, é comum também realizar-se alguma operação sobre os elementos examinados que não satisfazem a condição procurada.

Vamos estudar duas maneiras de fazer essa operação: uma usando os comandos `for` / `break` e outra usando `while`.

Busca linear com `for` / `break` Neste caso, o comando `for` se encarrega de fornecer os itens da lista, um a um, enquanto o `break` se encarrega de interromper a iteração ao se achar o item desejado.

```
item_ainda_não_encontrado = True
for item in lista:
    if item satisfaz a condição:
        item_ainda_não_encontrado = False
        break
    else:
        fazer alguma operação sobre o item
```

Busca linear com `while` Neste caso, o comando `while` deve se encarregar das duas tarefas: fornecer os itens da lista, um a um, enquanto o item não é encontrado e interromper a iteração quando isso acontecer.

```
item_ainda_não_encontrado = True
while item_ainda_não_encontrado and lista não esgotada:
    obter o próximo item da lista
    if item satisfaz a condição:
        item_ainda_não_encontrado = False
    else:
        fazer alguma operação sobre o item
```

1.1.12 Ordenar uma lista

A ordenação de uma lista pode ser feita pelo método `sort`, que ordena a lista no local, isto é, destroi a versão original da lista, ou pela função `sorted` que retorna uma nova lista com os valores na ordem desejada.

```
In [71]: nums = [1, 3, 4, -3, 7]
         'nums', id(nums), nums

Out[71]: ('nums', 4495772552, [1, 3, 4, -3, 7])

In [72]: nums.sort()
         'nums', id(nums), nums

Out[72]: ('nums', 4495772552, [-3, 1, 3, 4, 7])

In [73]: nums.sort(key=abs)
         'nums', id(nums), nums

Out[73]: ('nums', 4495772552, [1, -3, 3, 4, 7])

In [74]: nums.sort(reverse=True)
         'nums', id(nums), nums

Out[74]: ('nums', 4495772552, [7, 4, 3, 1, -3])

In [75]: pals = ['xis', 'alma', 'alfa', 'oi']
         'pals', id(pals), pals

Out[75]: ('pals', 4495732360, ['xis', 'alma', 'alfa', 'oi'])
```

```
In [76]: pals.sort()
         'pals', id(pals), pals

Out[76]: ('pals', 4495732360, ['alfa', 'alma', 'oi', 'xis'])

In [77]: pals.sort(key=len)
         'pals', id(pals), pals

Out[77]: ('pals', 4495732360, ['oi', 'xis', 'alfa', 'alma'])

In [78]: pals.sort(reverse=True)
         'pals', id(pals), pals

Out[78]: ('pals', 4495732360, ['xis', 'oi', 'alma', 'alfa'])

In [79]: nums1 = [1, 3, 4, -3, 7]
         'nums1', id(nums1), nums1

Out[79]: ('nums1', 4495731016, [1, 3, 4, -3, 7])

In [80]: nums2 = sorted(nums1)
         'nums2', id(nums2), nums2

Out[80]: ('nums2', 4495732616, [-3, 1, 3, 4, 7])

In [81]: nums2 = sorted(nums1, key=abs)
         'nums2', id(nums2), nums2

Out[81]: ('nums2', 4495732552, [1, 3, -3, 4, 7])

In [82]: nums2 = sorted(nums1, reverse=True)
         'nums2', id(nums2), nums2

Out[82]: ('nums2', 4495681992, [7, 4, 3, 1, -3])
```

1.1.13 Inverter a ordem dos itens de uma lista

```
In [83]: pals = ['xis', 'alma', 'alfa', 'oi']
         print('pals', id(pals), pals)

         pals.reverse()
         print('pals', id(pals), pals)

         revs = list(reversed(pals))
         print('revs', id(revs), revs)

pals 4495729032 ['xis', 'alma', 'alfa', 'oi']
pals 4495729032 ['oi', 'alfa', 'alma', 'xis']
revs 4495125000 ['xis', 'alma', 'alfa', 'oi']
```

1.1.14 Copiar uma lista

```
In [84]: anums = [1, 3, 4, -3, 7]

         bnums = anums.copy()
         'anums', id(anums), anums
         'bnums', id(bnums), bnums

Out[84]: ('anums', 4495123336, [1, 3, 4, -3, 7])

Out[84]: ('bnums', 4495125768, [1, 3, 4, -3, 7])

In [85]: cnums = list(anums)
         print('anums', id(anums), anums)
         print('cnums', id(cnums), cnums)

anums 4495123336 [1, 3, 4, -3, 7]
cnums 4495771656 [1, 3, 4, -3, 7]
```

*** ***Muito cuidado com aliasing*** *** Chama-se *aliasing* a situação em que mais do que uma variável encontra-se associada a um certo objeto, o que permite que esse objeto seja acessado de mais do que uma maneira.

É importante lembrar que, em Python, uma variável comporta-se como um rótulo que é colocado em um objeto mas pode ser transferido para outro a qualquer momento. Esse modelo contrasta com a visão tradicional de variável como sendo um contentor de dados de um determinado tipo.

Por exemplo, considere o código abaixo

```
In [86]: a = [1, 2, 3]
         b = a
```

A linha 1 cria uma lista de inteiros e associa o rótulo `a` a ela.

A linha 2 pega o objeto ao qual o rótulo `a` está associado e associa o rótulo `b` a ele.

Daí em diante, podemos nos referir a esse objeto usando o nome `a` ou o nome `b`.

Sabemos que todo objeto em Python possui um identificador único. A função `id` retorna o identificador do objeto associado a um certo nome. O comando abaixo mostra que os nomes `a` e `b` estão associados ao mesmo objeto.

```
In [87]: id(a), id(b)

Out[87]: (4495615240, 4495615240)
```

Python dispõe de dois operadores que nos ajudarão nesta discussão:

- `x == y` é avaliada como `True` se os objetos associados às variáveis `x` e `y` tiverem o mesmo valor.
- `x is y` é avaliada como `True` se as variáveis `x` e `y` estiverem associadas a um mesmo objeto.

No nosso exemplo, como `a` e `b` estão associados ao mesmo objeto, quando aplicados a eles, os operadores `==` e `is` devem retornar `True`.

```
In [88]: a == b
```

```
Out[88]: True

In [89]: a is b

Out[89]: True
```

O mesmo não acontece quando uma variável `c` é criada a partir de uma operação ralizada sobre `a`. Por exemplo, o comando abaixo cria uma cópia do objeto associado à variável `a` e associa a variável `c` a ela.

```
In [90]: c = list(a)
         c

Out[90]: [1, 2, 3]
```

Como consequência, `a` e `c` estão associadas a objetos distintos, mas que têm o mesmo valor.

```
In [91]: a == c

Out[91]: True

In [92]: a is c

Out[92]: False
```

Vamos agora modificar os valores dos objetos associados às variáveis `b` e `c`.

```
In [93]: b[1] = 20
         b

Out[93]: [1, 20, 3]

In [94]: c[2] = 30
         c

Out[94]: [1, 2, 30]
```

O que você acha que aconteceu com `a`?

```
In [95]: a

Out[95]: [1, 20, 3]
```

Como `a` é um *alias* de `b`, isto é, é um outro nome para um mesmo objeto, ele reflete as alterações que esse objeto sofreu.

Por outro lado, como `a` e `c` se referem a objetos distintos, o que acontece com um não interfere na vida do outro.

Moral da história

> Ao criar uma variável, veja se você não está criando um *alias* quando imaginava estar criando uma cópia.
>
> E se você precisar mesmo de um *alias* não se esqueça de que ele será afetado por todas as alterações sofridas pelo seu 'gêmeo'.

1.1.15 Slicing (*fatiamento*)

A operação de fatiamento (*slicing*) permite selecionar uma fatia (*slice*) com mais do que um elemento de uma lista.

Como no caso de `range`, *slicing* também admite três parâmetros não obrigatórios:
`umaLista[start:stop:step]`
Nesse caso, serão selecionados os elementos contidos numa faixa que inclui `start` mas não inclui `stop`, escolhidos de `step` em `step`, isto é,
`umaLista[start]`, `umaLista[start+step]`, `umaLista[start+2*step]`, `...` sem incluir ou ultrapassar `umaLista[stop]`.

```
In [96]: # Vamos criar uma lista numérica
         nums = [11, 22, 23, 34, 45, 16]
         print('nums', id(nums), nums)

nums 4495044040 [11, 22, 23, 34, 45, 16]
```

```
In [97]: # Seleção de uma faixa com todos os parâmetros
         imps = nums[0:6:2]
         print('nums', id(nums), nums)
         print('imps', id(imps), imps)

nums 4495044040 [11, 22, 23, 34, 45, 16]
imps 4495773320 [11, 23, 45]
```

```
In [98]: # Quando omitido, step assume o valor 1
         fatia = nums[1:5]
         print('nums ', id(nums), nums)
         print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4495553736 [22, 23, 34, 45]
```

```
In [99]: # Quando omitido, start assume o valor 0
         fatia = nums[:5]
         print('nums ', id(nums), nums)
         print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4495040840 [11, 22, 23, 34, 45]
```

```
In [100]: # Quando omitido, stop assume o valor len(lista)
          fatia = nums[1:]
          print('nums ', id(nums), nums)
          print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4494915592 [22, 23, 34, 45, 16]
```

```
In [101]: # Quando todos os parâmetros são omitidos, obtemos uma cópia da lista
          # Note que os ids são diferentes
          fatia = nums[:]
          print('nums ', id(nums), nums)
          print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4495772936 [11, 22, 23, 34, 45, 16]


In [102]: # Step pode ser negativo e, nesse caso, a relação entre start e stop se inverte
          fatia = nums[5:1:-1]
          print('nums ', id(nums), nums)
          print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4494915592 [16, 45, 34, 23]
```

Você consegue explicar bem este último resultado?
Você consegue antecipar o resultado de `fatia = nums[::-1]`?

```
In [103]: # Quando step é negativo e start e stop são omitidos, obtemos uma cópia invertida d
          fatia = nums[::-1]
          print('nums ', id(nums), nums)
          print('fatia', id(fatia), fatia)

nums  4495044040 [11, 22, 23, 34, 45, 16]
fatia 4495772936 [16, 45, 34, 23, 22, 11]
```

1.2 Conversão de listas para estruturas e de estruturas para listas

1.2.1 Conversão de *string* para *lista de strings*

Já vimos que *split* converte uma *string* em uma lista de *strings*, que pode depois, se desejado, ser convertida em uma lista de outro tipo.

```
In [104]: snums = '12  3   456 78   90'
          print('snums', type(snums), id(snums), repr(snums))

snums <class 'str'> 4495645912 '12  3   456 78   90'


In [105]: lnums = snums.split()
          print('lnums', type(lnums), id(lnums), lnums)

lnums <class 'list'> 4495772680 ['12', '3', '456', '78', '90']


In [106]: inums = [int(n) for n in lnums]
          print('inums', type(inums), id(inums), inums)

inums <class 'list'> 4495041736 [12, 3, 456, 78, 90]
```

1.2.2 Conversão de uma *lista de strings* para *string*

O método *join* faz essa conversão. A *string* sobre a qual se aplica o método é usada como separador entre os elementos da lista na *string* resultante.

```
In [107]: sres1 = ''.join(lnums)
          print('sres1', type(sres1), id(sres1), repr(sres1))
```

```
sres1 <class 'str'> 4495731440 '1234567890'
```

```
In [108]: sres2 = ' '.join(lnums)
          print('sres2', type(sres2), id(sres2), repr(sres2))
```

```
sres2 <class 'str'> 4495779824 '12 3 456 78 90'
```

```
In [109]: sres3 = '-x-'.join(lnums)
          print('sres3', type(sres3), id(sres3), repr(sres3))
```

```
sres3 <class 'str'> 4495019944 '12-x-3-x-456-x-78-x-90'
```

1.2.3 Conversão de uma *lista* qualquer para *string*

Quando os elementos da lista não forem *strings,* é necessário convertê-los antes de aplicar o método *join*.

```
In [110]: inums = [12, 3, 4.56, 78, 9.0]
          print('inums', type(inums), id(inums), inums)

          knums = [str(x) for x in inums]
          print('knums', type(knums), id(knums), knums)

          sres4 = ' '.join(knums)
          print('sres4', type(sres4), id(sres4), repr(sres4))
```

```
inums <class 'list'> 4495615304 [12, 3, 4.56, 78, 9.0]
knums <class 'list'> 4495041736 ['12', '3', '4.56', '78', '9.0']
sres4 <class 'str'> 4495604520 '12 3 4.56 78 9.0'
```

1.2.4 Dividir uma lista em pedaços de mesmo tamanho

Há várias maneiras de dividir uma lista em pedaços do mesmo tamanho. Uma delas usa o conceito de *list comprehension* que foi visto na *Aula09*.

```
In [111]: n = 20
          nums = [x for x in range(1, n + 1)]
          print('nums', nums)

          p = 5
          peds = [nums[i:i+p] for i in range(0, n, p)]
          print('peds', peds)
```

```
nums [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
peds [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20]]
```

1.2.5 Achatar (*flatten*) uma lista

O achatamento de uma lista converte uma *lista de listas* em uma lista simples e também pode ser implementado por uma *list comprehension*.

```
In [112]: lista = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19,
          flat = [item for sublista in lista for item in sublista]
          print('flat', flat)

flat [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Esse processo "achata"apenas um nível. Se os elementos da lista original forem também *listas de listas* a operação poderá ser repetida até chegar a uma lista completamente "achatada". Caso o aninhamento seja heterogêneo, será necessária uma abordagem mais potente, a ser desenvolvida nas *Aulas 25-27*.

```
In [113]: lista = [[1, 2], [3, 4, 5]], [[6, 7], [8], [9, 10]], [[11, 12, 13], [14, 15], [16,
          flat1 = [item for sublista in lista for item in sublista]
          print('flat1', flat1)

flat1 [[1, 2], [3, 4, 5], [6, 7], [8], [9, 10], [11, 12, 13], [14, 15], [16, 17, 18, 19, 20]]
```

```
In [114]: flat2 = [item for sublista in flat1 for item in sublista]
          print('flat2', flat2)

flat2 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

1.3 Exemplos de aplicação

1.3.1 Criar uma lista com os elementos comuns a outras duas listas

```
In [115]: list1 = [2 * x for x in range(10)]
          print('list1', id(list1), list1)

          list2 = [3 * x for x in range(10)]
          print('list2', id(list2), list2)

          inter = [x for x in list1 if x in list2]
          print('inter', id(inter), inter)

list1 4495615496 [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
list2 4495681480 [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
inter 4495773512 [0, 6, 12, 18]
```

1.3.2 Cálculo da média ponderada

Dadas uma lista de notas e uma lista de pesos, calcular a média ponderada das notas dadas.

```
In [116]: from random import choice

          pesos = [choice(range(1, 4)) for _ in range(10)]
          print('pesos', id(pesos), pesos)

          notas = [choice(range(11)) for _ in range(10)]
          print('notas', id(notas), notas)

          total = 0
          for i in range(min(len(notas), len(pesos))):
              total += notas[i] * pesos[i]

          media = round(total / sum(pesos), 1)
          print('total', total, 'média', media)
```

```
pesos 4495773640 [1, 2, 2, 3, 1, 3, 1, 3, 3, 3]
notas 4495126088 [4, 2, 5, 4, 3, 3, 0, 7, 2, 8]
total 93 média 4.2
```

1.3.3 Eliminar elementos repetidos de uma lista

Dada uma lista criar uma outra eliminando todos os elementos repetidos na primeira.

```
In [117]: import random

          repets = random.choices(range(1, 7), k=20)
          print('repets', id(repets), repets)

          unicos = []
          for x in repets:
              if x not in unicos:
                  unicos += [x]

          print('repets', id(repets), repets)
          print('unicos', id(unicos), unicos)

          sorted_unicos = sorted(unicos)
          print('sorted(unicos)', id(sorted_unicos), sorted_unicos)

          unicos.sort()
          print('unicos', id(unicos), unicos)
```

```
repets 4493688264 [5, 1, 2, 6, 4, 3, 2, 1, 2, 3, 2, 4, 4, 4, 5, 5, 4, 5, 2, 5]
repets 4493688264 [5, 1, 2, 6, 4, 3, 2, 1, 2, 3, 2, 4, 4, 4, 5, 5, 4, 5, 2, 5]
unicos 4495689992 [5, 1, 2, 6, 4, 3]
sorted(unicos) 4495882632 [1, 2, 3, 4, 5, 6]
unicos 4495689992 [1, 2, 3, 4, 5, 6]
```

1.3.4 Remover a pontuação de uma frase

Dada uma frase, remover todos os sinais de pontuação.

```
In [119]: import string

          frase = list(input('Digite uma frase: '))
          pontuação = list('!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~')

          frase_mod = []
          for c in frase:
              if c not in pontuação:
                  frase_mod += [c]
          frase_mod = ''.join(frase_mod)
          frase_mod
```

```
Out[119]: 'Socorramme subi no ônibus em Marrocos'
```

1.3.5 Remover a acentuação de uma frase

Dada uma frase, remover todos os sinais de acentuação.

```
In [121]: import string

          com_acentos = list('áàãâéêíóõôúçÁÀÃÂÉÊÍÓÕÔÚÇ')
          sem_acentos = list('aaaaeeioooucAAAAEEIOOOUC')


          frase = list(input('Digite uma frase: '))

          frase_mod = []
          for c in frase:
              if c in com_acentos:
                  frase_mod += sem_acentos[com_acentos.index(c)]
              else:
                  frase_mod += [c]
          frase_mod = ''.join(frase_mod)
          frase_mod
```

```
Out[121]: 'Socorram-me, subi no onibus em Marrocos!'
```

1.3.6 Trocar maiúsculas por minúsculas em uma frase

Dada uma frase, trocar todas as letras maiúsculas por minúsculas.

```
In [13]: import string

         maiúsculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZÁÀÃÂÉÊÍÓÕÔÚÇ')
         minúsculas = list('abcdefghijklmnopqrstuvwxyzáàãâéêíóõôúç')


         frase = list(input('Digite uma frase: '))

         frase_mod = []
         for c in frase:
             if c in maiúsculas:
                 frase_mod += minúsculas[maiúsculas.index(c)]
```

```
            else:
                frase_mod += [c]
        frase_mod = ''.join(frase_mod)
        frase_mod
```

```
Out[13]: 'socorram-me, subi no ônibus em marrocos!'
```

1.3.7 Verificar se uma frase é palíndroma

Dada uma frase, verificar se ela é palíndroma, desconsiderando maiúsculas/minúsculas, acentos, espaços e pontuação. Uma frase é palíndroma se ela puder ser lida igualmente nos dois sentidos.

Um esboço de solução com alto nível de abstração poderia ser:

- ler a frase
- eliminar caracteres a serem desconsiderados
- verificar se é palíndroma
- exibir o resultado da verificação

```
In [122]: import string
```

```
In [124]: # ler a frase original e convertê-la em uma lista
          frase_ori = input('Digite uma frase: ')
          frase_lista = list(frase_ori)
```

```
In [125]: # criar frase modificada, eliminando caracteres a serem desconsiderados
          minúsculas = list('abcdefghijklmnopqrstuvwxyzáàãâéêíóõôúç')
          maiúsculas = list('ABCDEFGHIJKLMNOPQRSTUVWXYZÁÀÃÂÉÊÍÓÕÔÚÇ')
          com_acentos = list('áàãâéêíóõôúçÁÀÃÂÉÊÍÓÕÔÚÇ')
          sem_acentos = list('aaaaeeioooucAAAAEEIOOOUC')
          pontuação = [' '] + list('!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~')

          frase_mod = []
          for c in frase_lista:
              if c in pontuação:
                  pass
              elif c in maiúsculas:
                  frase_mod += minúsculas[maiúsculas.index(c)]
              elif c in com_acentos:
                  frase_mod += sem_acentos[com_acentos.index(c)]
              else:
                  frase_mod += [c]
```

```
In [126]: # verificar se a frase modificada é palíndroma
          frase_eh_palindroma = True
          for i in range(len(frase_mod) // 2):
              if frase_mod[i] != frase_mod[-(i + 1)]:
                  frase_eh_palindroma = False
                  break
```

Uma implementação análoga mas usando um comando `while` seria

In [127]:
```
# verificar se a frase modificada é palíndroma
frase_eh_palíndroma = True
i = 0
while frase_eh_palíndroma and i < len(frase_mod) // 2:
    if frase_mod[i] != frase_mod[-(i + 1)]:
        eh_palíndroma = False
    else:
        i += 1
```

In [128]:
```
# exibir o resultado da verificação
if frase_eh_palindroma:
    print("'" + frase_ori + "'", 'é palíndroma.')
else:
    print("'" + frase_ori + "'", 'não é palíndroma.')
```

```
'Socorram-me, subi no ônibus em Marrocos!' é palíndroma.
```