Programação I / Introdução à Programação

Capítulo 5, "Data types" (listas em compreensão, dicionários)

João Pedro Pedroso

2024/2025

Última aula:

- listas → sequências com elementos de tipo arbitrário, mutáveis
- tuplos → como listas, mas imutáveis

```
>>> nota = ('Pedro', 12)
>>> nota[0]
'Pedro'
>>> nota[1]
12
>>> nota[0] = 'Joao'
TypeError: 'tuple' object does not support item assignment
```

Hoje:

- Listas em compreensão
- Dicionários

Resolução de exercícios: Soma de certos múltiplos

Implemente a função `sum_mult_k(j, k, n)` que calcula a soma dos inteiros de 1 a `n` (inclusive) que verificam as seguintes condições:

- são múltiplos de `j` mas não de `k`, ou
- são múltiplos de `k` mas não de `j`

Por exemplo, o resultado de `sum_mult_k(3,2,8)` é 17.

Nota: use parentesis na expressão lógica se tiver dúvidas em relação à precedência.

Resolução de exercícios: Soma de certos múltiplos

Implemente a função `sum_mult_k(j, k, n)` que calcula a soma dos inteiros de 1 a `n` (inclusive) que verificam as seguintes condições:

- são múltiplos de `j` mas não de `k`, ou
- são múltiplos de `k` mas não de `j`

Por exemplo, o resultado de `sum_mult_k(3,2,8)` é 17.

Nota: use parentesis na expressão lógica se tiver dúvidas em relação à precedência.

```
def sum_mult_k(j, k, n):
    '''Soma de certos múltiplos.'''
    total = 0
    for i in range(1,n+1):
        if (i % j == 0 and i % k != 0) or (i % k == 0 and i % j != 0):
            total += i
    return total
```

Resolução de exercícios: Soma de certos múltiplos

Implemente a função `sum_mult_k(j, k, n)` que calcula a soma dos inteiros de 1 a `n` (inclusive) que verificam as seguintes condições:

- são múltiplos de `j` mas não de `k`, ou
- são múltiplos de `k` mas não de `j`

Por exemplo, o resultado de `sum_mult_k(3,2,8)` é 17.

Nota: use parentesis na expressão lógica se tiver dúvidas em relação à precedência.

```
def sum_mult_k(j, k, n):
    '''Soma de certos múltiplos.'''
    total = 0
    for i in range(1,n+1):
        if (i % j == 0 and i % k != 0) or (i % k == 0 and i % j != 0):
            total += i
    return total
```

- Como testar?

```
1  >>> sum_mult_k(3, 9, 16)
2  36
```

Listas em compreensão

- É muito comum construir uma lista partindo de uma outra:
  - selecionando elementos usando uma condição;
  - aplicando uma transformação a cada elemento.
- Exemplo: calcular quadrados
  - construir a lista dos quadrados dos números inteiros de 1 a 9.
  - solução usando um ciclo for:

```
sqrs = []
for x in range(1, 10):
    sqrs.append(x**2)
print(sqrs)
```

Listas em compreensão

- É muito comum construir uma lista partindo de uma outra:
  - selecionando elementos usando uma condição;
  - aplicando uma transformação a cada elemento.
- Exemplo: calcular quadrados
  - construir a lista dos quadrados dos números inteiros de 1 a 9.
  - solução usando um ciclo for:

```
sqrs = []
for x in range(1, 10):
    sqrs.append(x**2)
print(sqrs)
```

  - solução usando uma lista em compreensão:

```
>>> sqrs = [x**2 for x in range(1,10)]
>>> sqrs
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Listas em compreensão

- Sintaxe: notação inspirada na teoria de conjuntos

$$\{x^2 : x \in \{1, \ldots, 9\}\}$$

  - em Python:

```
>>> sqrs = [x**2 for x in range(1,10)]
>>> sqrs
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

  - mais geralmente:

    `[<EXPRESSION> for <VARIABLE> in <SEQUENCE>]`

Mais exemplos

```
>>> [i**2 for i in [2,3,5,7]]
[4, 9, 25, 49]

>>> [1+x/2 for x in [0, 1, 2]]
[1.0, 1.5, 2.0]

>>> [ord(c) for c in "ABCDEF"]
[65, 66, 67, 68, 69, 70]

>>> [len(s) for s in
     "As armas e os barões assinalados".split()]
[2, 5, 1, 2, 6, 11]
```

Listas em compreensão com condições

- Exemplo: quadrados dos múltiplos de 3 inferiores a 10.

```
[x**2 for x in range(10) if x%3==0]
```

- Mais geralmente:

```
[<EXPRESSION> for <VARIABLE> in <SEQUENCE> if <CONDITION>]
```

Mais um exemplo

- Um número natural *n* é *primo* se não tem divisores próprios
  - *divisores próprios:* maiores do que 1 e menores do que *n*
- Para testar se *n* é primo podemos testar se a lista dos divisores próprios é vazia.
- Testar primos:

```
1  def primo(n):
2      # lista dos divisores próprios
3      divs = [d for d in range(2,n) if n%d==0]
4      # n é primo se e só se a lista for vazia
5      return len(divs)==0
```

- *Nota:* esta solução é ineficiente (calcula *todos* os divisores em vez de terminar após encontrar o primeiro...)

Listas em compreensão imbricadas

- Podemos usar uma lista em compreensão dentro de outra. Exemplo:

```
>>> [[i*j for j in range(1,11)] for i in range(1,11)]
```

produz a matriz da multiplicação:

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
 [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
...
 [9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]]
```

Compreensões com múltiplas sequências

- A ordem do resultado depende da ordem das sequências:

```
>>> [(x,y) for x in "ABC" for y in range(3)]
[('A', 0), ('A', 1), ('A', 2),
 ('B', 0), ('B', 1), ('B', 2),
 ('C', 0), ('C', 1), ('C', 2)]
>>> [(x,y) for y in range(3) for x in "ABC"]
[('A', 0), ('B', 0), ('C', 0),
 ('A', 1), ('B', 1), ('C', 1),
 ('A', 2), ('B', 2), ('C', 2)]
```

Sintaxe geral

```
[<EXPR> for <VAR1> in <SEQ1> if <COND1>
        for <VAR2> in <SEQ2> if <COND2>
        ...
        for <VARN> in <SEQN> if <CONDN>]
```

Exemplo:

```
>>> [(x,y) for x in "ABCDE" if x!= "C" for y in range(10) if y%3==0]
[('A', 0), ('A', 3), ('A', 6), ('A', 9),
 ('B', 0), ('B', 3), ('B', 6), ('B', 9),
 ('D', 0), ('D', 3), ('D', 6), ('D', 9),
 ('E', 0), ('E', 3), ('E', 6), ('E', 9)]
```

Dicionários

Motivação: um inventário

- Pretendemos associar *quantidades disponíveis* a *frutas*.

Item	Quantidade
bananas	25
peras	12
laranjas	10

- Poderíamos representar como uma lista de pares:

```
[('bananas',25), ('laranjas',10), ('peras',12) ]
```

- Problemas:
  - necessário percorrer a lista para procurar o valor associado a uma chave
  - permite duplicar chaves; por exemplo

```
[('bananas',1), ('bananas',25), ('laranjas',10), ('peras',12),]
```

Representa 1, 25 ou 26 bananas?

Dicionários

- Estrutura de dados para tabelas de associações
- Cada chave é associada a um (e um só) valor
- Analogia: dicionário bilingue (e.g. português-inglês)
- Podemos usar como chave:
  - números
  - cadeias de caracteres
  - tuplos
  - combinações dos anteriores

  (apenas *tipos imutáveis*)
- A ordem dos pares (chave, valor) é a ordem de inserção
  - *(a partir da versão 3.5 do Python)*

Utilização de dicionários

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
```

- Podemos consultar valores apartir da chave:

```
>>> inv['bananas']
25
>>> inv['bananas'] = inv['bananas'] + 1
>>> inv
{'bananas': 26, 'laranjas': 10, 'peras': 12}
```

- Mais alguns exemplos:

```
>>> emails = { 'Maria João':'mj@mail.pt',
               'João Pedro':'jp@mail.pt' }
>>> dirs = { 'N':(0,1), 'S':(0,-1),
             'W':(-1,0), 'E':(1,0) }
>>> vazio = {}      # inicializar um dicionário vazio
```

Algumas operações sobre dicionários

- Construção (*por extensão*):

  `mydict = {}` `yourdict = {<KEY>:<VALUE>, ...}`

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
```

- Acesso `[]` : obter valor associado a chave *(erro se não existir)*

  `<DICT>[<KEY>]`

```
>>> inv['bananas']
25
```

- Associar um valor a uma chave

  `<DICT>[<KEY>] = <VALUE>`

```
>>> inv['cerejas'] = 50
>>> inv['peras'] += 10
```

- Testar existência de uma chave *(resultado: booleano)*

  `<KEY> in <DICT>`

```
>>> 'cerejas' in inv, 'ameixas' in inv
(True, False)
```

Exemplos

```
>>> inv = {'bananas':25, 'laranjas':10, 'peras':12}
>>> inv['bananas']
25
>>> inv['kiwis']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'kiwis'
>>> 'bananas' in inv
True
>>> 'kiwis' in inv
False
>>> inv['cerejas'] = 50
>>> inv['peras'] += 10
>>> inv
{'bananas': 25, 'laranjas': 10, 'peras': 22, 'cerejas': 50}
```

Valores por omissão

- Obter o valor ou *default* se a *chave* não existir:

  `<DICT>.get(<KEY>, <DEFAULT>)`

- Exemplos:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> inv.get('bananas',0)
25
>>> inv.get('kiwis',0)
0
```

- Percorrer todas as chaves:

```
for <KEY> in <DICT>:
    ....
```

Percorrer dicionários

- Forma mais habitual:

```
>>> inv = {'laranjas':10, 'peras':12, 'bananas':25}
>>> for k in inv:
    ...:     print(k, inv[k])

laranjas 10
peras 12
bananas 25
```

- Percorrer por valores ordenados da chave: sorted

```
>>> for k in sorted(inv):
...     print(k, inv[k])
...
bananas 25
laranjas 10
peras 12
```

- Notas:
  - em versões antigas do Python as chaves não estão ordenadas
  - na versão 3.7 e posteriores: *ordem de inserção* é preservada

Dicionários: acesso a chaves e valores (1)

- Obter uma lista com todas as chaves: `<DICT>.keys()`

```
>>> inv = {'laranjas':10, 'peras':12, 'bananas':25}
>>> list(inv)
['laranjas', 'peras', 'bananas']
>>> list(inv.keys())   # outra forma
['laranjas', 'peras', 'bananas']
```

- Obter uma lista com todos os valores: `<DICT>.values()`

```
>>> list(inv.values())
[10, 12, 25]
```

Dicionários: acesso a chaves e valores (2)

- Lista das chaves e valores: método `<DICT>.items()`

```
>>> inv = {'laranjas':10, 'peras':12, 'bananas':25}
>>> inv.items()
dict_items([('laranjas', 10), ('peras', 12), ('bananas', 25)])
```

  - ver semelhança com `enumerate`

- Percorrer todos os pares de chaves e valores:

```
>>> for (k,v) in inv.items():
...     print(k,v)
...
laranjas 10
peras 12
bananas 25
```

Em resumo...

- Dicionários permitem representar tabelas de associações
- Ordem entre as entradas: depende da versão do Python
  - anteriores a 3.7 $\rightarrow$ ordem não está definida
  - 3.7 e posteriores: $\rightarrow$ ordem da sequência de inserção
- Pesquisa pela chave é mais eficiente do que sobre uma lista de pares

Noções estudadas

Nesta aula

- listas em compreensão
- dicionários

Próxima aula

- dicionários: exemplos, conclusão