# Programação I / Introdução à Programação

*Apêndice B, "More datatypes"*

João Pedro Pedroso

2024/2025

## Última aula

- Ficheiros: conceito
- Ficheiros de texto
- Folhas de cálculo: ficheiros CSV
- Ficheiros para objetos Python: Pickle, JSON

## Hoje:

- Conjuntos (‘*set*‘s) e mais tipos de dados em Python
- Módulo `collections` da biblioteca standard

# Tipos de dados "nativos" em Python

- Já vimos:
  - `int / float / bool`
  - `str`
  - `list / tuple`
  - `dict`
- Hoje: `set` $\rightarrow$ conjuntos

# Tipos mutáveis e imutáveis

- Mutáveis: listas, dicionários

```
>>> my_list = [2, 4, 5, 3, 6, 1]
>>> my_list[0] = 9
>>> my_list
[9, 4, 5, 3, 6, 1]
```

- Imutáveis: *strings*, tuplos

```
>>> my_tuple = (2, 5, 3, 1)
>>> my_tuple[0] = 9
Traceback (most recent call last):
  File "<interactive input>", line 2, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

# Problemas com objetos mutáveis

```
>>> list_one = [1, 2, 3, 4, 6]
>>> list_two = list_one
>>> list_two[-1] = 5
>>> list_one
[1, 2, 3, 4, 5]
```

`list_one` e `list_two` são duas referências para o mesmo lugar na memória:

```
>>> id(list_one) == id(list_two)
True
```

# Forma de ultrapassar este problema

Fazendo cópia do objeto:

```
>>> list_one = [1, 2, 3, 4, 6]
>>> list_two = list(list_one)
>>> id(list_one) == id(list_two)
False
>>> list_two[-1] = 5
>>> list_two
[1, 2, 3, 4, 5]
>>> list_one
[1, 2, 3, 4, 6]
```

- para listas (ou outras estruturas) recursivas:
  - módulo copy → `copy.deepcopy()`

# Conjuntos em Python: `sets`

- `set` → coleção sem ordem definida, sem elementos duplicados
- Utilizações:
  - testar pertença (para coleções grandes, é muito mais rápido do que com listas)
  - eliminação de duplicados
  - operações matemáticas sobre conjuntos:
    - união: `a | b` → elementos em `a` ou em `b`
    - interseção: `a & b` → elementos em `a` e em `b`
    - complemento (diferença): `a - b` → elementos em `a` mas não em `b`
    - diferença simétrica: `a ^ b` → elementos em `a` ou em `b`, mas não em ambos
- Criar conjuntos em Python:

```
a = set()       # empty set
b = {1,2,3}     # set initialized from sequence of values
c = set([1,2,3])  # initialize from list
```

# Exemplos:

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # fast membership testing
True
>>> 'crabgrass' in basket
False

>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> b
{'l', 'a', 'z', 'c', 'm'}
>>> a - b                              # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                              # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # letters in both a and b
{'a', 'c'}
>>> a ^ b                              # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

# Atualização de conjuntos

```
>> my_set = set([1, 4, 2, 3, 4])
>>> my_set
{1, 2, 3, 4}
>>> my_set.add(13)    # add elements to set
>>> my_set
{1, 2, 3, 4, 13}
>>> my_set.remove(1)    # remove elements from set
>>> my_set
{2, 3, 4, 13}
>>> my_set.remove(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
>>> my_set
{2, 3, 4, 13}
>>> my_set.discard(2)    # remove if included, otherwise do nothing
>>> my_set
{3, 4, 13}
>>> my_set.discard(2)    # no error here
>>> my_set
{3, 4, 13}
>>>
```

# Elementos de conjuntos e mutabilidade

- Elementos de conjuntos (`set`) devem ser imutáveis
- Idem para chaves de dicionários

```
>>> a = [1,2,3]
>>> s = {1, 2, a}
Traceback (most recent call last):
  File "<python-input-1>", line 1, in <module>
    s = {1, 2, a}
        ^^^^^^^^^
TypeError: unhashable type: 'list'
>>> a = (1,2,3)
>>> s = {1, 2, a}
>>> s
{1, 2, (1, 2, 3)}
```

## Quadro: listas e conjuntos

| | Ordenada | Não ordenada |
|---|---|---|
| Mutável | `list` | `set` |
| Imutável | `tuple` | `frozenset` |

# Exemplo: listas como chaves em dicionários

- Por vezes precisamos de associar valores a listas
- Primeira ideia: criar dicionário em que a chave é uma lista

```
>>> d = {}
>>> a = [1,2,3]
>>> fa = 322
>>> d[a] = fa
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>>
```

- O valor das chaves tem de ser imutável
  - converter lista para tuplo

```
>>> d[tuple(a)] = fa
>>> d
{(1, 2, 3): 322}
>>> d[1,2,3]
322
```

# Exemplo: conjuntos com chaves em dicionários

- Também não podemos criar dicionários em que a chave é um conjunto

```
>>> d = {}
>>> a = {1,2,3}
>>> fa = 322
>>> d[a] = fa
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>>
```

- $\rightarrow$ converter `set` para `frozenset`

```
>>> d[frozenset(a)] = fa
>>> d
{frozenset({1, 2, 3}): 322}
```

# Módulo `collections` – *Container datatypes*

`https://docs.python.org/3/library/collections.html#collections.namedtuple`

Seleção de alguns tipos:

| | |
|---|---|
| namedtuple() | factory function for creating tuple subclasses with named fields |
| deque | list-like container with fast appends and pops on either end |
| Counter | dict subclass for counting hashable objects |
| defaultdict | dict subclass that calls a factory function to supply missing values |

# namedtuple

- atribuem significado a cada posição num tuplo
- tuplos ficam mais legíveis
- permitem aceder a registos por nome (em vez de posição)

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)     # instantiate with positional or keyword arguments
>>> p[0] + p[1]             # indexable like the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__ with a name=value style
Point(x=11, y=22)
```

# deque

- Generalização de *stacks* e *queues*
- Como listas, mas permite append e pop no fim e no início

```
>>> from collections import deque
>>> d = deque('ghi')                 # make a new deque with three items
>>> for elem in d:                   # iterate over the deque's elements
...     print(elem.upper())
G
H
I
>>> d.append('j')                    # add a new entry to the right side
>>> d.appendleft('f')                # add a new entry to the left side
>>> d                                # show the representation of the deque
deque(['f', 'g', 'h', 'i', 'j'])
>>> d.pop()                          # return and remove the rightmost item
'j'
>>> d.popleft()                      # return and remove the leftmost item
'f'
```

# Counter

## Contador de objetos

```
>>> from collections import Counter
>>> cnt = Counter()
>>> for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
...     cnt[word] += 1
>>> cnt
Counter({'blue': 3, 'red': 2, 'green': 1})
```

- `elements()` → itera sobre os elementos guardados, ignorando contagens negativas

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> sorted(c.elements())
['a', 'a', 'a', 'a', 'b', 'b']
```

- `most_common([n])` → retorna `n` tuplos com os elementos mais frequentes

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
```

# defaultdict

- Zero como valor por omissão

```
>>> from collections import defaultdict
>>> d = defaultdict(int)
>>> d[7] += 123
>>> d
defaultdict(<class 'int'>, {7: 123})
```

  - default_factory → int

- Exemplo: listas como valores em dicionários

```
>>> from collections import defaultdict
>>> d = defaultdict(list)
>>> d[2].append("abc")
>>> d
defaultdict(<class 'list'>, {2: ['abc']})
```

  - default_factory → list

## Aula de hoje:

- Mais tipos de dados do Python:
  - conjuntos (`set`, `fronzenset`)
- Módulo `collections`

## Próximas aulas

- Programação orientada a objetos