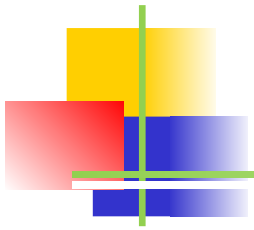


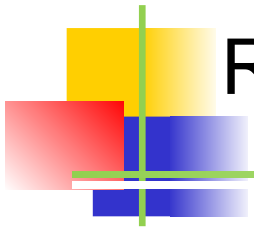
Fundamentos de Programação

António J. R. Neves

João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática

Universidade de Aveiro

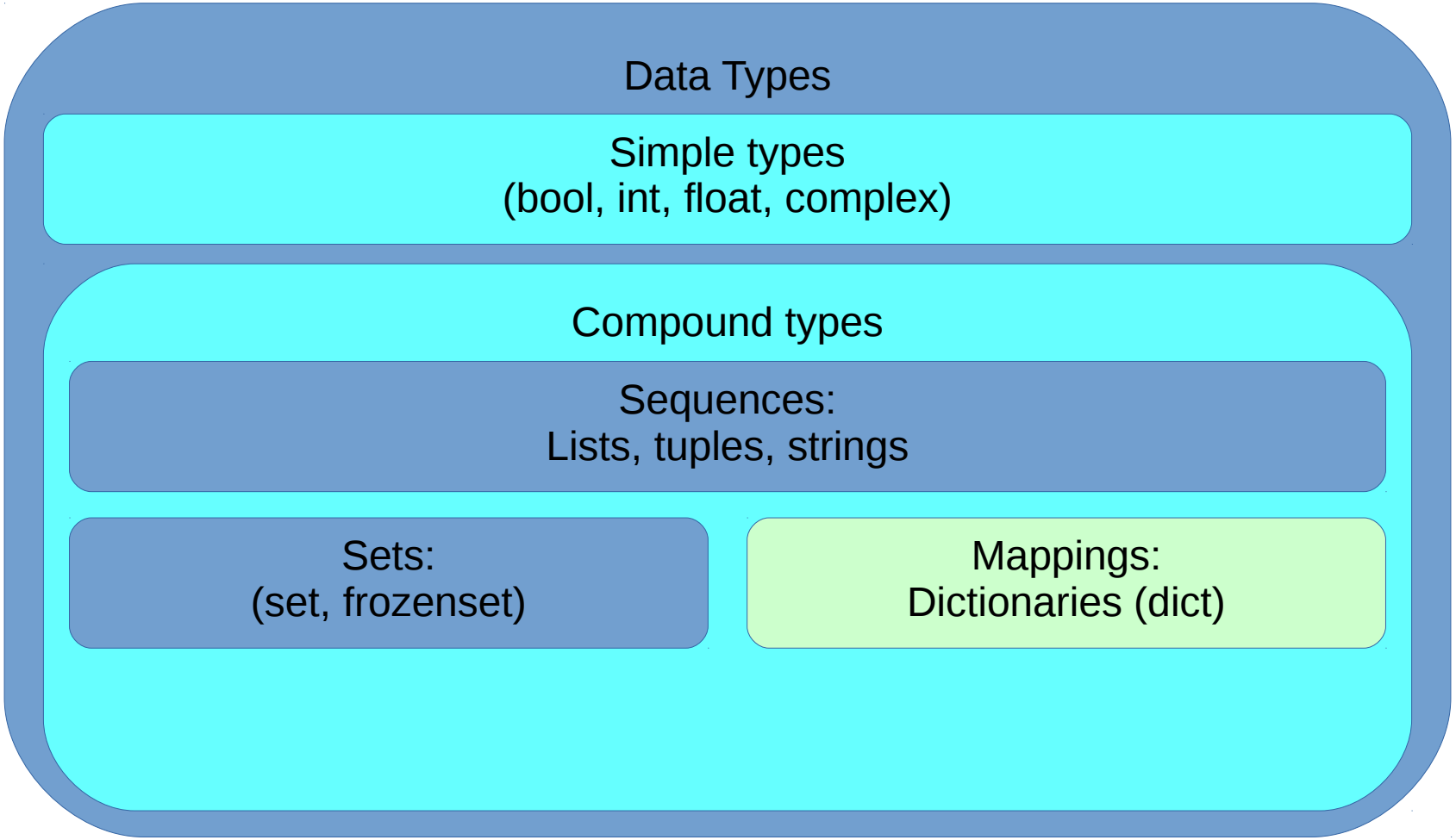


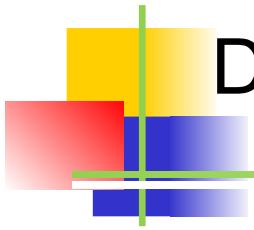
Resumo

- Dictionaries

Dictionaries

- Python data types





Dictionaries

- A **dictionary** is an *unordered*, *associative collection* of items.
  - A **collection** because they may contain zero or more items.
  - **Associative** because each item associates a **key** to a **value**.
  - **Unordered** because, unlike lists or strings, items are not in sequence from first to last.
- Dictionaries are also called **associative arrays** or **maps**.
  - Because they establish a *mapping* between keys and values.
- Dictionary items are also called **key-value pairs**.
- A dictionary may be created using braces (curly brackets).

```
eng2sp={'one': 'uno', 'two': 'dos', 'three': 'tres'}
shop = {'eggs': 12, 'sugar': 1.0, 'coffee': 3}
```

- An empty dictionary may be created with `{}` or `dict()`.

Dictionaries: accessing items

- To access the value for a given key, use square brackets.

```
shop['sugar']     #-> 1.0
eng2sp['two']     #-> 'dos'
```

- Dictionaries are **mutable**.

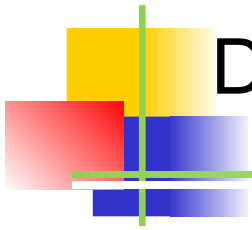
```
shop['eggs'] = 24   # Change the value for a key
shop['bread'] = 6   # Add a new key-value association
```

- **Values** in a dictionary can be of any type.

```
shop['eggs'] = [1, 'a']
shop['eggs'] = {'brown': 6, 'white': [2, 3]}
```

- **Keys** may be ints, floats, strings, tuples or essentially any other immutable type. So, lists are not allowed!

```
eng2sp[4] = 'quatro'   # integer key is fine
shop[[1,2]]            #-> TypeError
```



Dictionaries *versus* lists

- In a sense, a dictionary is a kind of list, but more general. In a list, the indices are integers. In a dictionary, keys can be any type of object (almost).
- However, the order of items in a dictionary is unpredictable.

```
>>> d = {10: 'dez', 20: 'vinte', 1000: 'mil'}
>>> print(d)
{1000: 'mil', 10: 'dez', 20: 'vinte'}
```

- Also, you cannot take slices from dictionaries!

```
d[10:20]     #-> TypeError
```

Dictionary methods

- The `len` function returns the number of key-value pairs.
- The `in` operator tells you whether something appears as a key in the dictionary. (This is efficient!)

```
'two' in eng2sp        #-> True ('two' is a key)
'uno' in eng2sp        #-> False ('uno' is not a key)
```

- Three methods return sequences of keys, values and items.

```
d.keys()   #-> [1000, 10, 20]
d.values() #-> ['mil', 'dez', 'vinte']
d.items()  #-> [(1000, 'mil'), (10, 'dez'), (20, 'vinte')]
```

- So, to see whether something is a value in the dictionary, you could use:

```
>>> 'uno' in eng2sp.values()
True
```

Dictionary methods (2)

- Trying to access an inexistent key is an error.

```
d[10]     #-> 'dez'
d[33]     #-> KeyError
```

- But using the `get` method will return a default value.

```
d.get(10)           #-> 'dez'
d.get(33)           #-> None
d.get(33, 'oops')   #-> 'oops'
```

- We can delete an item with the `del` operator.

```
del d[20]
print(d)          #-> {1000: 'mil', 10: 'dez'}
```

- Or use `pop` to delete an item and return its value.

```
x = d.pop(10)   #-> x == 'dez'
print(d)        #-> {1000: 'mil'}
```

Dictionary traversal

- The `for` instruction may be used to traverse dictionary *keys*.

```
for k in shop:
    print(k, shop[k])
```

```
eggs 24
bread 6
sugar 1.0
coffee 3
```

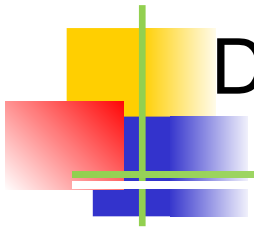
- This is equivalent to:

```
for k in shop.keys():
    print(k, shop[k])
```

- We may also traverse (key, value) pairs directly:

```
for k, v in shop.items():
    print(k, v)
```

Dictionaries: examples



- Suppose you are given a string and you want to count how many times each letter appears in some message:

```
d = dict()
for c in message:
    if c not in d:
        d[c] = 1
    else:
        d[c] += 1
```

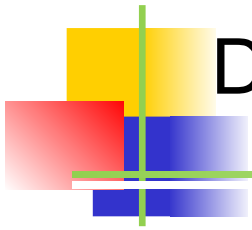
- If you use a dictionary in a `for` statement, it traverses the keys of the dictionary:

```
for c in d:
    print(c, d[c])
```

Dictionaries: examples (2)

- Create a dictionary that maps from frequencies to letters:

```
inverse = dict()
for key in d:
    val = d[key]
    if val not in inverse:
        inverse[val] = [key]
    else:
        inverse[val].append(key)
s = 'parrot'
print(d) # from previous slide
{'a': 1, 'p': 1, 'r': 2, 't': 1, 'o': 1}
print(inverse)
{1: ['a', 'p', 't', 'o'], 2: ['r']}
```



Dictionaries and lists of tuples

- Dictionaries have a method called `items` that returns a sequence of tuples, where each tuple is a key-value pair.

```
d = {'a':0, 'b':1, 'c':2}
t = d.items()  #→ dict_items(('a', 0), ('c', 2), ('b', 1))
```

- We can use a list of tuples to initialize a new dictionary:

```
t = [('a', 0), ('c', 2), ('b', 1)]
d = dict(t)      #→ {'a': 0, 'c': 2, 'b': 1}
```

- Combining items, tuple assignment and for:

```
for key, val in d.items():
    print(val, key)
```