Fundamentos de Programação

António J. R. Neves
João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Resumo

- Dictionaries

Dictionaries

Python data types

Data Types
Simple types   (bool, int, float, complex)
Compound types (collections)
Sequences: (list, tuple, str)
Sets: (set, frozenset)   **Mappings:** Dictionaries (dict)

Dictionaries

- A **dictionary** is an *unordered collection* of *unique*, *associative* items.

  - **Collection** because it may contain zero or more items.

  - **Unordered** because items are <u>not in sequence</u> from first to last.

  - **Associative** because each item <u>associates</u> a **key** to a **value**.

  - **Unique** because no two items can have the same key.

- 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**.

Dictionaries: creating and accessing items

- 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()`.

- 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
```

Dictionaries: value and key types

- **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 <u>immutable</u> objects. So, lists are not valid keys!

```
eng2sp[4] = 'quatro'         # integer key is fine
d[[1,2]] = 'A'               #-> TypeError: unhashable type
d[(12,25)] = 'Christmas'     # tuple key is fine
```

- Actually, keys must be <u>*hashable*</u>. In practice, this means:

  - keys must be immutable scalars, or

  - immutable collections containing only hashable elements.

```
d[(1,[2,3])] = 'quatro'     #-> TypeError: unhashable type
```

Dictionaries *versus* lists

- When accessing items, a dictionary is a kind of generalized list. In a list, the indices are integers. In a dictionary, keys can be any type of object (almost).

```
lst = [50, 51, 52]
dic = {'um':1, 'vinte':20, 'mil':1000}
lst[1]        #-> 51
dic['mil']    #-> 1000
```

- However, the order of items in a dictionary is irrelevant and unpredictable.

```
{'a':1, 'b':2} == {'b':2, 'a':1}           #-> True
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 <u>fast</u>!)

```
'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 (but this is <u>slow</u>):

```
'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*.

```
shop = {'eggs':24, 'bread':6,
         'coffee':3, 'sugar':1.0}
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:

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

▶ll

- To show the results, traverse the keys with a **for** statement:

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

Dictionaries: examples (2)

- Create a dictionary that maps from frequencies to letters:

```python
inverse = dict()
for key in d:
    val = d[key]
    if val not in inverse:
        inverse[val] = [key]
    else:
        inverse[val].append(key)

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: updating

- Many applications require updating a dictionary one item at a time.

- This can be done in several alternative, but equivalent ways.

- Example: counting characters in a message.

```
#A
d = {}
for c in message:
    if c not in d:
        d[c] = 1
    else:
        d[c] += 1
```

```
#B
d = {}
for c in message:
    if c not in d:
        d[c] = 0
    d[c] += 1
```

```
#C
d = {}
for c in message:
    d[c] = d.get(c, 0) + 1
```

```
#D
d = {}
for c in message:
    d.setdefault(c, 0)
    d[c] += 1
```

Dictionaries: updating (2)

- Example: grouping words in lists according to word length.

```
#A
d = {}
for w in wordlist:
    k = len(w)
    if k not in d:
        d[k] = [w]
    else:
        d[k].append(w)
```

```
#B
d = {}
for w in wordlist:
    k = len(w)
    if k not in d:
        d[k] = []
    d[k].append(w)
```

```
#C
d = {}
for w in wordlist:
    k = len(w)
    d[k] = d.get(k, [])
    d[k].append(w)
```

```
#D
d = {}
for w in wordlist:
    k = len(w)
    d.setdefault(k, []).append(w)
```

```
wordlist=['to','be','or','not','to','be','that','is','the','question']
#A, B, C or D...
d -> {8: ['question'], 2: ['to', 'be', 'or', 'to', 'be', 'is'], 3:
['not', 'the'], 4: ['that']}
```

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)
```