Fundamentos de Programação

António J. R. Neves João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática Universidade de Aveiro

Summary

- List comprehensions
- Dictionary and set comprehensions.
- Generator expressions.

Building lists

- Quite often, we need to <u>create collections</u> with elements related to those in <u>another collection</u>.
- For example: create a list of the squares of the values in lst.

- Another example: create a list of uppercase versions of the strings in lst.
 - What do you need to change in the code above?
- These programs always follow the <u>same</u> basic <u>pattern</u>.

List comprehensions

Python provides a more concise way to produce such lists.

- These are **list comprehensions**: <u>expressions</u> that generate lists by operating on the elements of other collections.
- Here the for..in clause is part of the expression. Do not confuse it with the for statement.

List comprehensions (2)

List comprehensions may also include if clauses.

```
args3 = [ s.upper() for s in args if len(s)>3 ]
#-> ['APPLE', 'DELL']
```

 List comprehensions may include multiple for..in and if clauses.

```
[(a,b) for a in [1,2] for b in nums if b>3*a]
#-> [(1, 4), (1, 7), (2, 7)]
```

Dictionary and set comprehensions

We may also create dictionaries by comprehension.

```
args = ['apple', 'dell', 'ibm', 'hp', 'sun']
d = { a: len(a) for a in args }
#-> {'apple': 5, 'ibm': 3, 'hp': 2, ...}
```

- Other variations are possible too, of course.
- Sets may also be defined by comprehension.

```
s = \{ 2+x \text{ for } x \text{ in } [3, 4, 5, 4] \}
```

Generator expressions

- Generator expressions are identical to the expressions used in list comprehensions, but enclosed in ().
- They create an object that generates items only <u>if and</u> <u>when needed</u>, unlike list comprehensions. This strategy is called <u>lazy evaluation</u> and can save memory and time.
- They're convenient as arguments to some functions.

```
nums = [4, -5, 3, 7, 2, 3, 1]

sum(x/2 for x in nums if x%2==0) #-> 3.0

all(x>0 for x in nums) #-> False
```

 We may use generator expressions to create other types of sequences, for example.

```
tuple( v for v in nums if v<3 ) #-> (-5, 2, 1)
```