

Fundamentos de Programação

António J. R. Neves João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

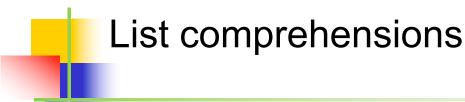
an@ua.pt / jmr@ua.pt
http://elearning.ua.pt/

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



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

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





 Python provides a different, more concise way to produce lists like these.

```
nums= [4, -5, 3, 7, 2, 3, 1]
nums2 = [ v**2 for v in nums ]
#-> [16, 25, 9, 49, 4, 9, 1]
args = ['apple', 'dell', 'ibm', 'hp', 'sun']
args2 = [ s.upper() for s in args ]
#-> ['APPLE', 'DELL', 'IBM', 'HP', 'SUN']
```

- These are **list comprehensions**: <u>expressions</u> that generate lists by operating on the elements of other collections.
- The for...in clause is part of the expression, not a 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]
#-> [(1, 4), (1, 7), (2, 4), (2, 7)]
```



Dictionary and set comprehensions

iniversidade de aveiro departamento de electrónica, elecomunicacões e informática

We may also create dictionaries by comprehension.

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

- Other variations are possible too, of course.
- Sets (we'll see them later) may also be defined by comprehension.

. . .



Generator expressions



- Generator expressions are identical to the expressions used in list comprehensions, but enclosed in ().
- They create an object that generates values only <u>if and</u> when needed, unlike list comprehensions.

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