



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 create lists with elements related to those in another list.
- For example: return a list of the squares of the values in `lst`.

```
lst = [1, -3, 2]
lst2 = []
for v in lst:
    v2 = v**2
    lst2.append(v2)
print( lst2 )
```

init result with empty list
loop over original list:
compute a new value
append it to result
#-> [1, 9, 4] [▶▶](#)

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



List comprehensions

- Python provides a more concise way to produce such lists.

```
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**: expressions that generate lists by operating on the elements of other collections.
- The **for...in** clause is part of the expression. It is 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

- 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.

```
s = { 2+x for x 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 *if and when needed*, unlike list comprehensions. This strategy is called *lazy evaluation*.
- 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)
```



Iterator objects?

- X...
- T