Programming Fundamentals

Data Types: Lists

João Correia Lopes

INESC TEC, FEUP

08 November 2018

GOALS

By the end of this class, the student should be able to:

- Describe the use of listas, which are sequences of elements of different types
- Enumerate the main methods available to work with lists

BIBLIOGRAPHY

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3, 2018 (Section 5.3) [PDF]
- Brad Miller and David Ranum, Learning with Python: Interactive Edition. Based on material by Jeffrey Elkner, Allen B. Downey, and Chris Meyers (Chapter 10) [HTML]
- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3 (RLE), 2012 (Chapter 11) [HTML]

TIPS

- There's no slides: we use a script and some illustrations in the class. That is NOT a replacement for **reading the bibliography** listed in the *class plan*
- "Students are responsible for anything that transpires during a class—therefore **if you're not in a class**, you should get notes from someone else (not the instructor)"—David Mayer
- The best thing to do is to **read carefully** and **understand** the documentation published in the Content wiki (or else **ask** in the class)
- We will be using **Moodle** as the primary means of communication

CONTENTS

A COMPOUND DATA TYPE

- So far we have seen built-in types like `int`, `float`, `bool`, `str` and we've seen lists, pairs or tuples
- Strings, **lists**, and tuples are qualitatively different from the others because they are made up of smaller pieces
- Lists group any number of items, of different types, into a single compound value
- Types that comprise smaller pieces are called **collection** or **compound data types**
- Depending on what we are doing, we may want to treat a compound data type as a single thing

LISTS

- A **list** is an ordered collection of values
- The values that make up a list are called its **elements**, or its **items**
- Lists and strings — and other collections that maintain the order of their items — are called **sequences**

LIST VALUES

- There are several ways to create a new list

```
1  numbers = [10, 20, 30, 40]
2
3  words = ["spam", "bungee", "swallow"]
4
5  stuffs = ["hello", 2.0, 5, [10, 20]]
```

- A list within another list is said to be **nested**
- a list with no elements is called an **empty list**, and is denoted `[]`

ACCESSING ELEMENTS

- The syntax for accessing the elements of a list is the index operator: `[]`
  - the syntax is the same as the syntax for accessing the characters of a string
- The expression inside the brackets specifies the index
- Remember that the indices start at 0
- Negative numbers represent reverse indexing

```
1  >>> numbers = [10, 20, 30, 40]
2
3  >>> numbers[1]
4  >>> numbers[-3]
```

⇒ https://github.com/fpro-admin/lectures/blob/master/12/lindex.py

LIST LENGTH

- The function `len` returns the length of a list, which is equal to the number of its elements
- It is a good idea to use this value as the upper bound of a loop, as it accommodates changes in the list

```
1  horsemen = ["war", "famine", "pestilence", "death"]
2
3  for i in range(len(horsemen)):
4      print(horsemen[i])
5
6  len(["car makers", 1, ["Ford", "Toyota", "BMW"], [1, 2, 3]])
```

LIST MEMBERSHIP

- `in` and `not in` are Boolean operators that test membership in a sequence

```
>>> horsemen = ["war", "famine", "pestilence", "death"]
>>> "pestilence" in horsemen
True
>>> "debauchery" in horsemen
False
>>> "debauchery" not in horsemen
True
```

⇒ https://github.com/fpro-admin/lectures/blob/master/12/students.py

LIST OPERATIONS

- The + operator concatenates lists:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6
```

- Similarly, the * operator repeats a list a given number of times:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

LIST SLICES

- The slice operations we saw previously with strings let us work with sublists:

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3]
['b', 'c']

>>> a_list[:4]
['a', 'b', 'c', 'd']

>>> a_list[3:]
['d', 'e', 'f']

>>> a_list[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

LISTS ARE MUTABLE

- Unlike strings, lists are mutable, which means we can change their elements
- An assignment to an element of a list is called **item assignment**

```
>>> fruit = ["banana", "apple", "quince"]
>>> fruit[0] = "pear"
>>> fruit[2] = "orange"
>>> fruit
['pear', 'apple', 'orange']
```

⇒ https://github.com/fpro-admin/lectures/blob/master/12/lassign.py

LIST DELETION

- Using slices to delete list elements can be error-prone
- The `del` statement removes an element from a list

```
>>> a = ["one", "two", "three"]
>>> del a[1]
>>> a
["one", "three"]
```

OBJECTS AND REFERENCES

- Since strings are *immutable*, Python optimizes resources by making two names that **refer** to the same string value refer to the same object

```
>>> a = "banana"
>>> b = "banana"
>>> a is b
True

>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True

>>> a is b
False
```

⇒ https://github.com/fpro-admin/lectures/blob/master/12/references.py

ALIASING

- Since variables refer to objects, if we assign one variable to another, both variables refer to the same object
- Although this behavior can be useful, it is sometimes unexpected or undesirable

```
>>> a = [1, 2, 3]
>>> b = a

>>> a is b
True

>>> b[0] = 5
>>> a
[5, 2, 3]
```

CLONING LISTS

- If we want to modify a list and also keep a copy of the original
- The easiest way to **clone** a list is to use the slice operator

```
>>> a = [1, 2, 3]
>>> b = a[:]      # considered bad practice
>>> c = a.copy() # better in Python3

>>> b
[1, 2, 3]

>>> b[0] = 5

>>> a
[1, 2, 3]
```

USING ZIP()

```
1   coordinate = ['x', 'y', 'z']
2   value = [3, 4, 5, 0, 9]
3
4   result = zip(coordinate, value)
5   resultList = list(result)
6   print(resultList)
7
8   c, v =  zip(*resultList)
9   print("c =", c)
10  print("v =", v)
```

⇒ https://github.com/fpro-admin/lectures/blob/master/12/zip.py
⇒ http://www.pythontutor.com/visualize.html

LIST OPERATIONS

- See the Python Standard Library for a comprehensive list of "Common Sequence Operations": PSL
- See the Python Standard Library for a comprehensive list of operations on "Mutable Sequence Types": PSL

EXERCISES

- Moodle activity at: LE12: Lists