Table of contents

2.7 Structured data types: lists and strings

Python includes structured data types. A variable of one of these types represents, with a single identifier or name, several values of basic/structured data types.

In this subject we are going to see:

- Lists → Sequences of elements of heterogeneous data types
- Strings → Sequences of characters

Lists

- A list is a sequence of elements of any type.
- The number of elements can change during the execution of the program → mutable data type.
- A list is represented by a sequence of items in between square brackets, separated with commas.

```
>>> my_list = ["smile", 15, 3.1416]
>>> type(my_list)
<type 'list'>
>>> len(my_list)
3
```

Lists: access to individual items

- An individual item of a list can be accessed using its index.
- Indices start by 0, which represents the first item in the list.

```
>>> my_list = ["smile", 15, 3.1416]
>>> my_list[0]
'smile'
>>> type(my_list[2])
<class 'float'>
>>> i = 1
>>> my_list[i]
15
```

Lists: operators

- Concatenation: +
- Repetition: *
- Slicing: [:]
- Comparison, membership, identity: == != in is

```
>>> p = [1, 2, 3, 4]
>>> print(p+p)
[1, 2, 3, 4, 1, 2, 3, 4]
>>> q=p*3
>>> print(q)
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> print(q[3:7])
[4, 1, 2, 3]
>>> print(3 in p and p!=q)
True
```

Slicing

It selects the items of a list from begin until end-1 using step

```
list[begin:end:step]
```

If we omit *begin* or *end*, the first or the last item are selected (both included).

If *step* is omitted, the default value is 1.

Negative values mean positions to count backwards from the end.

```
numbers = ["one", "two", "three", "four", "five"]

numbers[1:4]     ⇒     ['two', 'three', 'four’]
numbers[2:-1]    ⇒     ['three', 'four’]
numbers[:]       ⇒     ['one', 'two', 'three', 'four', 'five']
numbers[::2]     ⇒     ['one', 'three', 'five']
numbers[::-1]    ⇒     ['five', 'four', 'three', 'two', 'one']
```

Lists: functions

- Length of a list: `len()`
- Creation of numeric lists: `list(range())`
- Functions for numeric lists: `sum()` `max()` `min()`
- Sorting of items: `sorted()`

```
>>> p = list(range(1,5))
>>> print(p, len(p))
[1, 2, 3, 4] 4
>>> print(sum(p), max(p), min(p))
10 4 1
>>> print(sorted(p*3))
[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]
>>> names = ["Emily", "Mark", "John", "Sophie"]
>>> print(sorted(names))
['Emily', 'John', 'Mark', 'Sophie']
```

Lists: modifications

Lists are mutable variables. This means that we can modify the contents of a list without having to create a new one.

- Direct assignment of an item: p[1] = 7
- Insertion: .append() .extend() .insert()
- Deletion: .pop() .remove()

```
>>> numbers = ["one", "two", "three", "four", "five"]
>>> numbers[2] = "five"
>>> print(numbers)
['one', 'two', 'five', 'four', 'five']
>>> numbers.append("six")
>>> numbers.extend(["seven", "eight"])
>>> print(numbers)
['one','two','five','four','five','six','seven','eight']
```

Lists: modifications

```
>>> numbers.pop()
'eight'
>>> print(numbers)
['one', 'two', 'five', 'four', 'five', 'six', 'seven']
>>> numbers.pop(2)
'five'
>>> print(numbers)
['one', 'two', 'four', 'five', 'six', 'seven']
>>> numbers.remove("seven")
>>> print(numbers)
['one', 'two', 'four', 'five', 'six']
>>> numbers.insert(2, "three")
>>> print(numbers)
['one', 'two', 'three', 'four', 'five', 'six']
```

Lists: iteration over items

A **for** loop allows us to get the items of a list sequentially. It is also possible to iterate over all the elements using their indexes.

```
>>> p = [1, 2, 3, 5, 7, 11, 13, 17]
>>> for i in p:
...     print(i**2, end=' ')
...
1 4 9 25 49 121 169 289
>>>
>>> for i in range(len(p)):
...   print(p[i]**2, end=' ')
...
1 4 9 25 49 121 169 289
```

Strings

A text string is a special type of list, in which all the items are characters (digits, letters, symbols and special control characters like tabulators and line breaks) represented between double or single quotes.

They are immutable, thus they cannot be modified directly.

```
>>> name = "Patrick"
>>> type(name)
<class 'str'>
>>> print(name[2], len(name))
't’ 7
>>> name[2]="p"
TypeError: 'str' object does not support item assignment
```

Strings: special characters

Special characters are represented by a code after a **backslash**.

`\n`	⇒	line break
`\t`	⇒	tabulator
`\\`	⇒	backslash \
`\'` or `\"`	⇒	single or double quotes

```
>>> sentence = "first line \n second \t \" line \" "
>>> print(sentence)
first line
 second 	 " line "
>>> sentence
'first line \n second \t \" line \" '
```

Strings: operators

- The same used in lists: `+` `*` `[:]` `==` `!=`
- Membership: `in` (looking for characters or substrings)

```
>>> name = "Anthony"
>>> print("thon" in name)
True
>>> other_name = "Timot" + name[5:7]
>>> print(other_name)
'Timotny'
```

Strings: Functions and methods

- Length `len(cadena)`
- Keyboard reading `input()`
- Type conversion `int()` `float()` `str()`
- Methods `.join(sequence), .split(separator)`

```
>>> print (int("123") + float("12.3"), "123" + str(123))
135.3 123123
>>> id = "00000014-Z"
>>> number_letter = id.split("-")
>>> print(number_letter)
['00000014', 'Z']
>>> id_no_hyphen = "".join(number_letter)
>>> print(id_no_hyphen)
'00000014Z'
```

Lists of lists: matrices

- An item of a list can be another list.
- This feature allows us to build lists of lists.
- In this subject we will use them to build matrices.

```
>>> x=[[1.2,3.4,2.6],[0.1,30,5.5]]
>>> print(x[0])
[1.2, 3.4, 2.6]
>>> print(x[0][0])
1.2
>>> print(x[1][1])
30
>>> print(x[1][2])
5.5
```

Each item of x is a list of 3 numbers.

x[0] stores the first row.

x[0][0] is the first item of that row, the upper left corner of the matrix.

x[1][2] is the third item of the second row.

Lists of lists: matrices

To get all the items of a matrix we need two **nested loops**.

```
>>> for i in range(len(x)):
...   for j in range(len(x[i])):
...     print (x[i][j], end=' ')
...   print()
...
1.2 3.4 2.6
0.1 30 5.5
>>>
```

`len(x)` is the number of rows, so `i` will be 0 and 1.

`len(x[i])` is the number of cols, so `j` will be 0, 1, and 2.

The inside loop is executed first for i=0, so it is as we have written `x[0][j]` and, in the second iteration of the outside loop, `x[1][j]`. The value of `j` is first 0, then 1 and finally 2.

Therefore, the inside loop accesses to all the index pairs: `[0][0], [0][1], [0][2], [1][0], [1][1], [1][2]`

Lists of lists: matrices

- Functions operate with lists of lists as in the case of a simple list.
- See the two functions below:
  - The first one returns a matrix of zeros.
  - The second one adds two matrices and returns the result.

```
def zeros(rows,cols):
        a = []
        for i in range(rows):
                a.append([0]*cols)
        return a

def add_matrices(a,b):
        c = zeros(len(a), len(a[0]))
        for i in range(len(a)):
                for j in range(len(a[0])):
                        c[i][j] = a[i][j]+b[i][j]
        return c
```