Bocconi

COMPUTER SCIENCE

code 30424 a.y. 2020-2021

Lesson 13

Sequences and dictionaries

Università Bocconi
MILANO

ITEC - IT Education Center

Objectives of the lesson

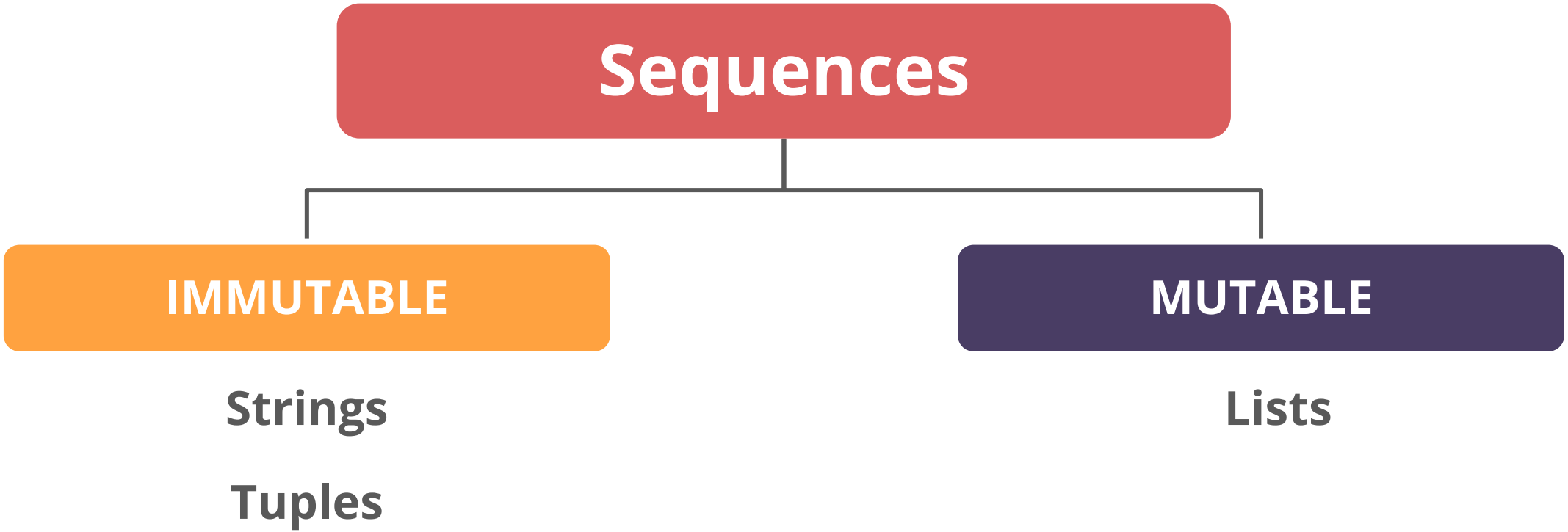
Learning how to access and handle:

- Strings
- Lists
- Tuples
- Dictionaries

Sequences

- Sequences are objects that hold multiple items of data, stored one after the other
- There are different types of sequences, including strings, lists and tuples
- Although they have different characteristics, sequences:
  - are iterable objects
  - can be mutable or immutable
  - the position of each value is identified by a number (index)
  - use many functions, methods and operations that allow you to access and work on the data

Mutability of sequences



Strings

- **A string** is a sequence of characters that can contain alphanumeric characters and symbols
- Strings are **immutable**, so we cannot change an existing string, but it's possible to create a new string that is a variation on the original

  Example:

```
>>> univ = 'Bocconi'
>>> univ = 'UniBocconi'
```

Lists

- A list is a sequence of values (elements) of any type, enclosed in square brackets and divided by commas:

  [19, 30, 24, 28] ['Bocconi', 'IULM', 'Bicocca'] [156, 'Via Roma', 'Milano', 3318488888, 15.36]

- Lists are assigned to **variables**:

  grades = [19, 30, 24, 28]

- A list with no elements is called an **empty list: [ ]**
- A list can be **nested** within another list : ['New York', 9.5, 2, [10, 20]]
- To create a list it is possible to use the built-in function **list**:

```
>>> list('Bocconi')
['B', 'o', 'c', 'c', 'o', 'n', 'i']
```

```
>>> list(range(4))
[0, 1, 2, 3]
```

Tuples

- A tuple is sequence very similar to a list. The elements of a tuple can be of any type, are comma-separated and enclosed in a set of parentheses :

```
>>> t = ('b', 'o', 'c', 'c', 'o', 'n', 'i')
```

- Although not necessary, it is common to enclose tuples in parentheses
- To create a tuple we can use the built-in function tuple

```
>>> tuple('Bocconi')
('B', 'o', 'c', 'c', 'o', 'n', 'i')
```

```
>>> tuple(range(4))
(0, 1, 2, 3)
```

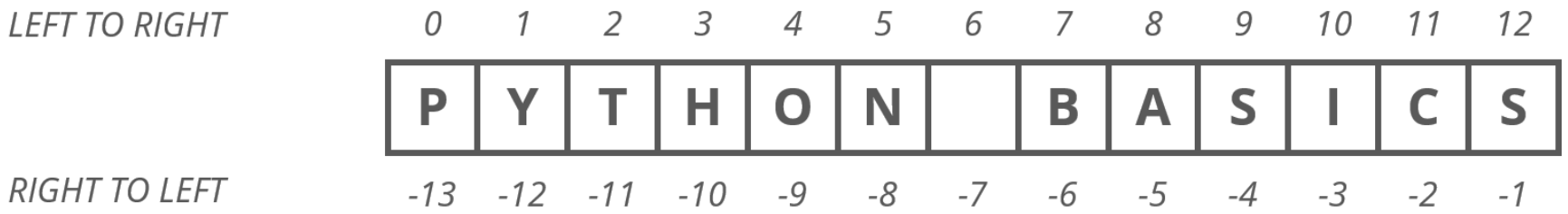
- Tuples are immutable, so it is not possible to modify an existing tuple: the only possibility is to create a new tuple, variant of the original.

Operators and functions of sequences

Operator/function	Description
+	Concatenates/merges two sequences
*	Creates more copies of a sequence and merges them
in	Returns *True* if an element is found in a sequence, otherwise it returns *False*
len(seq)	Returns the number of elements of the sequence
max(seq) and min(seq)	Returns the largest/smallest value in a sequence consisting only of strings or numbers
sorted(seq)	Returns a new list with the elements sorted in ascending order
sum(seq)	Sums the elements of the sequence (numbers only)

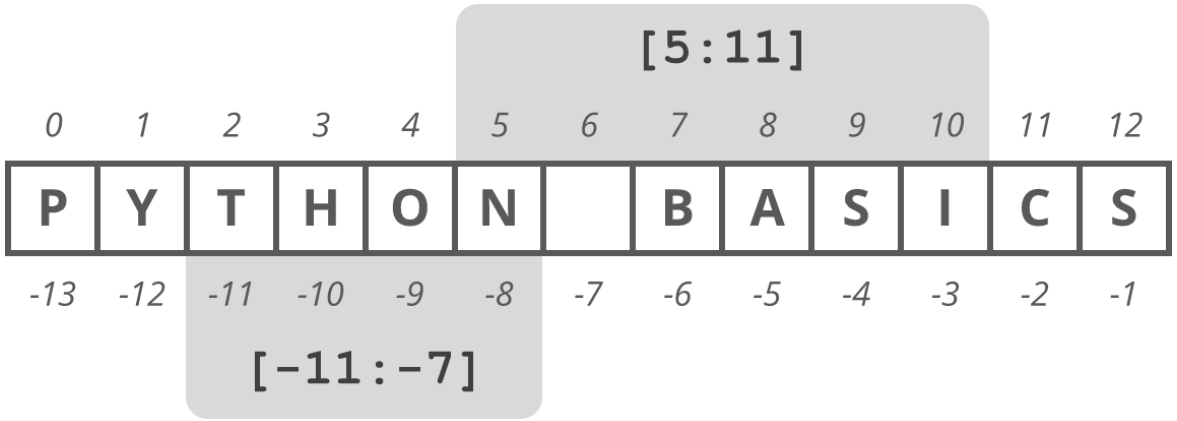
Indexing

- The position of each element within a sequence is identified by an integer, called an index
- Indexing allows access to the individual elements of a sequence, using the following syntax: **sequence[index]**
- The index of the first element on the left is equal to 0, while the index of the last element of the sequence is equal to the number of elements of the sequence minus 1
- It is possible to use indices with negative values: in this case the count starts from the end of the sequence



Slicing

- A segment or portion of a sequence is called a **slice**
- Slicing allows to select more than one element of a sequence using the syntax: **sequence[start:end:step]**
- The operation returns all the elements of the sequence from the one with the start index (included) and the one with the end index (not included). Step is optional and indicates which subsequent indexes to select after the first one



Examples of slicing

- When *start* is omitted, the slicing selects the elements starting from the first one; when *end* is omitted, the slicing selects the elements up to the last one. By omitting both, the entire sequence is selected
- Consider the string 'Computer viruses are an urban legend' stored in the variable *MyQuote*. Here are some examples of slicing:

Command	Meaning	Result
MyQuote[6:12]	Select all the elements from the one with index 6 to the one with index 12 (not included)	`'er vir'`
MyQuote[:7]	Select all the elements from the first one up to the one with index 7 (not included)	`'Compute'`
MyQuote[3:20:2]	Selects an element every two starting from the one with index 3 up to the one with index 20 (not included)	`'ptrvrssae'`
MyQuote[:15:3]	Selects an element every three starting from the first one	`'Cpevu'`

Methods

- Strings, lists and tuples have special functions, called **methods**
- The **syntax of functions** requires that the sequence or the name of the variable in which it is stored is one of the arguments of the function
- The **syntax of methods** requires that the name of the sequence is always specified, using the following syntax:

```
object.method(arguments)
```

Methods of strings

Method	Description
.upper()	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase is not modified
.lower()	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, is not modified
.find(sub)	Search in the string for the substring specified in the *sub* argument and returns the index of the first occurrence found
.startswith(prefix)	Returns *True* if the string starts with the string specified in the *prefix* argument, otherwise it returns *False*
.endswith(suffix)	Returns *True* if the string ends with the string specified in the *suffix* argument, otherwise it returns *False*
.count(sub)	Returns the number of occurrences of the substring *sub* in the string
.split()	Splits a string in words and returns a list of strings, considering the space character as the separator between the words
.join(iterable)	Returns a string obtained concatenating all the elements of an iterable object containing only strings

Methods of lists

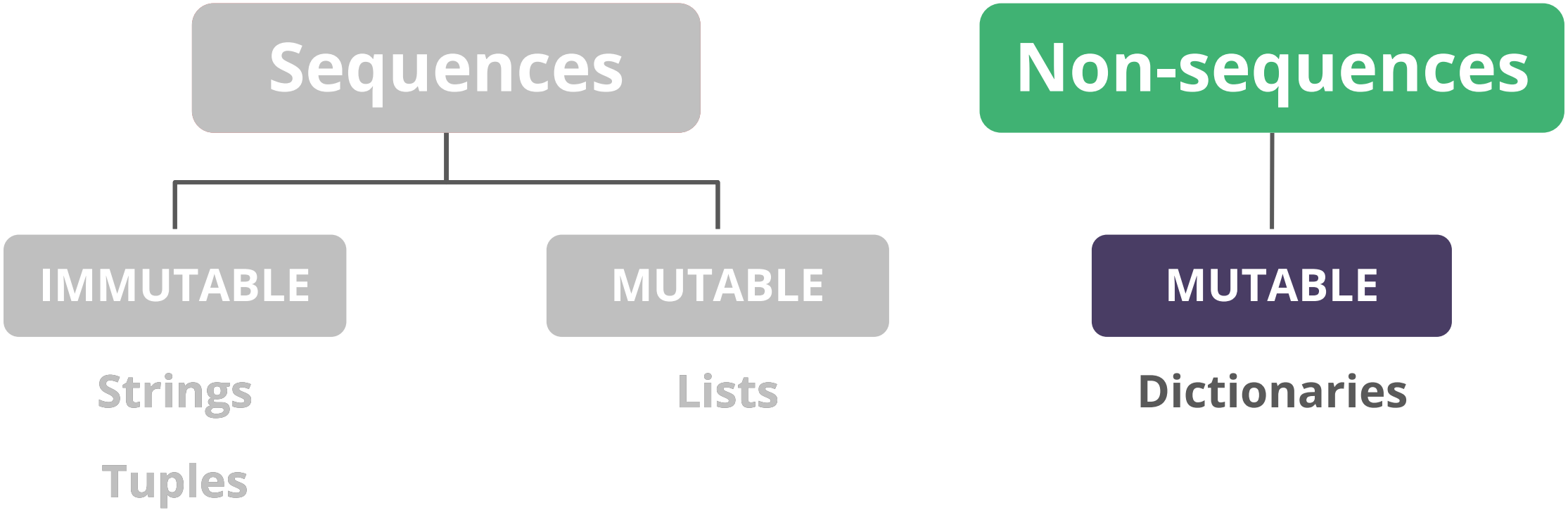
Method	Description
.append(element)	Appends new elements to the list
.insert(index, element)	Inserts the desired element in the position specified by the *index* parameter
.remove(element)	Searches for the specified element in the list and removes the first occurrence
.pop([index])	Removes and returns the element with the specified index. If we omit the index, it removes the last element of the list
.index(element)	Searches for the specified element within the list and returns its index
.count(element)	Counts how many times a given element is found in the list
.sort()	Sorts the elements of the list in ascending order
.reverse()	Reverses the order of the elements of the list

Methods of tuples

- The tuple methods are few because tuples are immutable
- It is often useful to convert a tuple into a list, work on the elements of the list and then convert it back into a tuple

Method	Description
.index(element)	Searches for the specified element within the tuple and returns its index. If the element occurs more than one time, it returns the index of the first occurrence
.count(element)	Returns the number of occurrences of a specified element in a tuple

Sequences -vs- Non-sequences



Dictionaries

- In Python, a **dictionary** is an object that contains a collection of data or items
- Element are made of two parts: a **key** and a **value**. The keys are unique and can be any kind of immutable objects. Values can be any kind of object, mutable or immutable.
- To create a dictionary it is necessary to write in **curly brackets { }** the elements divided by commas (,). Every element is made of a key followed by a colon (:) and a value
- Alternatively you can use the dict function, which creates a new dictionary with no elements
- Dictionaries are **mutable** objects

Keys and values

- The order of items in a dictionary is unpredictable, for this reason we cannot use indexing as we can do in sequences
- To retrieve a value stored in the dictionary we use the corresponding key :

```
dict_name[key]
```

- In order to add key-value pairs:

```
dict_name[key] = value
```

Operations with dictionaries

- in allows to check the presence of a key
- del removes a key-value pair from a dictionary
- len returns the number of key-value pairs

Methods of dictionaries

Method	Description
.get(key[,default])	Returns the value associated with the key specified in the *key* argument. If the specified key is not found in the dictionary, it returns the value *None* or the value specified in the *default* argument
.pop(key[,default])	Removes the key (and the associated value) specified in the key argument and returns the associated value. If the specified key is not found in the dictionary and a default value has not been set in the default argument, it returns a KeyError error
.popitem()	Removes the last key-value pair added to the dictionary and returns a tuple with two elements (the key and the value removed)
.items()	Creates a dict_items object consisting in a list of tuples, each one with two elements (the key-value pairs)
.keys()	Creates a dict_keys object consisting in a list in which each element is one of the keys of the dictionary
.values()	Creates a *dict_ values* object consisting in a list in which each element is one of the values of the dictionary

Traversing dictionaries

- The keys of a dictionary can be traversed using a for loop:

```
>>> address_book = { 'pippo': '555-123456', 'pluto': '555-654321' }
>>> for key in address_book:
        print(key)
```

- We can also traverse key-value pairs using the for loop of the list dict.items()

```
>>> for key, value in address_book.items():
        print(key, value)
```

Tools to learn with today's lecture

- How to handle and use strings
- How to handle and use lists
- How to handle and use tuples
- How to handle and use dictionaries

Files of the lesson

In the zip file **30424 ENG - 27 - Lesson 13.zip** you will find these files:

- **30424 lesson 13 slides.pdf**: these slides
- **Exercises 30424 lesson 13**: exercises on the content of the lesson

Book references

Learning Python:

Chapters 8 (except 8.12), 9

Assignment:

Exercises of lesson 13