CHAPTER 9 – TUPLES AND DICTIONARIES

9.1

Tuples are sequences, very similar to lists.

The elements of a tuple are enclosed in brackets (they can also not be) and are separated by a comma.

Tuples may contain data of any type, even non-uniform data.

Tuples are immutable.

Reasons to use tuples over lists:

* Safer
* Easier for Python to process

To convert tuples into lists use the list function:

Nameoflist = list (nameoftuple)

To convert lists into tuples use the tuple function:

Nameoftuple = tuple (nameoflist)

Indexing and slicing can be used on tuples

9.2

The main operators for tuples are:

+ 🡪 concatenates two tuples in a new tuple

\* 🡪 creates more copies of a tuple and merges them

in 🡪 returns True if an element is included in a tuple, False otherwise

not in 🡪 opposite of in

The main functions for tuples are:

len() 🡪 returns the length of a tuple

tuple() 🡪 converts an iterable object into a tuple

max() 🡪 returns the element with the highest value of a tuple

min() 🡪 returns the element with the lowest value of a tuple

sorted() 🡪 returns a new list with the tuple elements sorted in ascending order

sum() 🡪 returns the sum of all the elements of the tuple

The two tuple methods are:

.index(element) 🡪 returns the index of the first occurrence of an element

.count(element) 🡪 returns the number of occurrences of an element in a tuple

9.3

Tuples can be traversed by using indexes and slicing with loops

9.4

A dictionary is an object that stores a collection of data.

Dictionaries are unsorted and they are not sequences (i.e. indexing can’t be used)

Each element of a dictionary has two parts: a unique key (any immutable object) and a value (any object)

To create a dictionary we must enclose its elements in a set of curly braces and separate them with a comma:

Dictionaryname = { key1 : value1 , key2 : value2 , … }

If two or more identical keys are inserted, the last one overwrites the others

The syntax to add a new key value pair to a dictionary is:

Dictionaryname [newkey] = newvalue

To access the values stored in a dictionary, we use the corresponding key:

Dictionaryname [key]

To modify the value associated to a key the syntax is similar:

Dictionaryname [key] = [newvalue]

To delete a dictionary use the del function:

del dictionaryname

To delete a key value pair from a dictionary use the del function:

del dictionaryname [key]

9.5

The logical operators for dictionaries are:

in 🡪 returns True if a key is found in a dictionary, False otherwise

not in 🡪 opposite of in

is 🡪 returns True if a dictionary is a copy of another (i.e. if a change is made in one dictionary it is also applied to the other), False otherwise (the == operator can be used instead)

is not 🡪 opposite of is

The main functions for dictionaries are:

len() 🡪 returns the number of key value pairs

dict() 🡪 creates an empty dictionary or uses the argument to create a dictionary if the values in the argument have already been paired together

max()🡪 returns the maximum key value

min() 🡪 returns the minimum key value

sorted() 🡪 returns a list with the keys of a dictionary sorted in ascending order

The main methods for dictionaries are:

.clear()🡪empties a dictionary

.get(key[, default]) 🡪 returns the value associated with the key, if the key is not found and default is not specified it returns None, otherwise it returns the specified default value.

.pop(key[, default]) 🡪 removes the key value pair specified and returns the value, if the key is not found and default is not specified it returns KeyError, otherwise it returns the specified default value.

.popitem() 🡪 it removes the last key value pair of the dictionary and returns the key value pair as a tuple

.update(dict2) 🡪 adds the key value pairs of dict2 to the dictionary

.items()🡪 creates a dict\_items object consisting in a list of tuples each made of a key value pair -- it is a view object

.keys() 🡪 creates a dict\_keys list of all keys -- it is a view object

.values() 🡪 creates a dict\_values list of all values -- it is a view object