PYTHON FOR DATA SCIENCE DATA STRUCTURES

Ana-María Dobre

based on EDX Course

SEP 2023

1. LISTS & TUPLES (I)

Lists and tuples are called COMPOUND DATA TYPES.

TUPLES are an ordered sequence.
They are written as comma-separated elements, within parenthesis: (1)

A tuple can contain different types of data: (2)

Each elem of a tuple can be accessed via an index: (3)

In Python, you can also use **negative index:** (4)

We can **concatenate** or combine tuples by adding them: (5)

Example of **SLICING** tuples (if you would like to retrieve multiple elements from a tuple): (6)
Note: the last index is one larger than the index of the last element you want.

Use the len command to get the length: (7)

A tuple is **IMMUTABLE** i.e. we can't change it.
As a consequence of immutability, if we want to change a tuple, we must create a new tuple instead. For example, if we want to sort a tuple, we use the function sorted: (8)

NESTING: A tuple can contain other tuples, as well as other complex data types: (9)

```
(1) tuple1 = (10, 10, 7, 8, 9, 10, 10, 8)

(2) tuple2 = ('geronimo', 10, 2.0)

(3) tuple2[0] => "geronimo"

(4) tuple2[-1] => 2.0
tuple2[-2] => 10
tuple2[-3] => "geronimo"

(5) tuple3 = tuple2 + ("Mel", 10)
=> ('geronimo, 10, 2.0, "Mel", 10)

(6) tuple3[0:3]
=> ('geronimo', 10, 2.0) // retrieves first 3 elements

tuple3[3:5]
=> ("Mel", 10) // retrieves last 2 elements

(7) len(('geronimo', 10, 2.0)) => 3

(8) sortedTuple = sorted(tuple1)
// Note: the input is the original tuple.
The output is a new sorted list:
[7, 8, 8, 9, 10, 10, 10, 10]

(9)
NT = (10, 20, ("la oreja de Van Gogh", "rosas"), (8, 9), ("drums", (1, 2, 3))
NT[2] => ("la oreja de Van Gogh", "rosas")

Note: We can visualise this tuple as a tree, where we can access a deeper level of the tree by adding another set of square brackets:
NT[2][1] => "rosas"
NT[2][1][0] => "r"
```

2. LISTS & TUPLES (II)

Lists are also an ordered sequence.
They are written as comma-separated elements, within square brackets: (10)

In many respects, lists are like tuples.
One key difference is that lists are **MUTABLE**. For example, we can change the first element: (11)

NESTING: We can nest other lists, as well as tuples and other data structures: (12)

Access via index examples: (13) // incl. negative index

Example of SLICING in lists (if you would like to retrieve multiple elements from a list): (14)
Notes:
- the last index is one larger than the index of the last element you want
- **the index conventions for lists and tuples are identical**

We can concatenate or combine the lists by adding them: (15), which is the same as using keyword **"extend"**

If we apply the **append** method instead of extend, we add one more element to the list: (16)

Delete an element of a list by using the **del command**. We simply indicate the list item we would like to remove as an argument: (17)

Convert a string to a list using **split**: (18)
Note: If the delimiter is not the default one (empty space), it should be passed as an argument.

ALIASING: When we set one variable B equal to A, both A and B are referencing the same list.
Multiple names referring to the same object is aka aliasing: (19)

You can **clone list A** by using syntax: (20)

We can get more info on lists, tuples, and many other objects in Python using the help command: (21)

```
(10) list1= ['geronimo', 10, 2.0]

(11) list1[0] = "I call it Geronimo"

(12)  list2= ['geronimo', 10, 2.0, [2022, 2023], ('Mel', 100)]

(13) list2[0] => 'geronimo'
list2[3] => [2022, 2023]
list2[-1] => ("Mel", 100)

(14) list2[3:5]
=> [[2022, 2023], ('Mel', 100)]

(15) L1 = ["geronimo", 10, 2.0]
L2 = L1 + ["drums", 15]
// same as:
L2 = L1.extend(["drums", 15])
=> ["geronimo", 10, 2.0, "drums", 15]

(16) L2 = L1.append(["drums", 15])
=> ["geronimo", 10, 2.0, ["drums", 15]]

(17) del(list1[0]) => [10, 2.0]
del(list1[1]) => ['geronimo', 2.0]

(18) "el privilegio de amarte".split()
=> ["el", "privilegio", "de", "amarte"]

"a,b,c,d".split(",") => ["a", "b", "c", "d"]

(19) A = ["la oreja", 10, 2.0]
B = A
Side effect if we make change: A[0] = "Van Gogh"
=> The value of B[0] will change as a consequence.

(20) B = A[:]

(21) help(A)
```

3. DICTIONARIES

A dictionary has KEYS and VALUES.
A KEY is analogues to a list's INDEX. It is like an address, but it does not have to be an integer. It is often a string.

A dictionary is denoted with curly brackets.

The keys have to be immutable and unique.
The values can be immutable (e.g. a tuple), **mutable** (e.g. a list) **and duplicates** (e.g. an integer): (22)

Each key and value pair is separated by a comma.

Use square brackets with key as argument to output the value: (23)

Add a **new entry to the dictionary** as follows: (24)

Delete entry in the dictionary as follows: (25)

Use the method keys to get all keys: (26)
Note: the output is a list-like object with all the keys.

In the same way, we can obtain the values using the method values: (27)

```
(22) dict =
{"key1": 1, "key2": "2", "key3":  [30, 30, 30], "key4": (40, 40, 40), "key5": 5}

(23)
dict["key1"] => 1

(24)
dict["key6"] = "Shamanic drums"

(25)
del(dict["key6"]))

(26)
dict.keys()
=> ["key1", ... , "key6"]

(27)
dict.values()
```

4. SETS

SETS are a type of collection => like lists and tuples, you can input different Python types.

Unlike lists and tuples, they are UNORDERED.

SETS only have unique elements => there is only one of a particular element in a set.

To define a set, use curly brackets: (28)
Note: When the actual set is created, duplicate items will not be present.

Convert a list to a set by using the **function set()**: (29)

Add a new element to the set with **add**: (30)

Remove an element from the set with **remove**: (31)

Verify if an element is in the set with **in**: (32)

Intersection of two sets with **ampersand**: (33)

Union of two sets with **union**: (34)

Check if a set is a subset with issubset method: (35)

```
(28)
Set1 = { "Luciano", "Jose", "Placido", "Luciano", "Alessandro", "Placido"}
=> { "Luciano", "Jose", "Placido", "Alessandro"}

(29)
album = ["safina", "luna", "luna", "il mirto e la rosa", 2000]
album_set = set(album)
=> {"safina", "luna", "il mirto e la rosa", 2000}

(30)
album_set.add("siane")

(31)
album_set.remove("siane")

(32)
"luna" in album_set => True
"Patrick" in album_set => False

(33)
A = { "Sean", "Patrick", "John", "Jim" }
B = { "Patrick", "Calabazo", "Jim"}
C = A & B
=> { "Patrick" , "Jim"}

(34)
A.union(B)
=> { "Sean", "Patrick", "John", "Jim", "Calabazo" }

(35)
B.issubset(A) => False
```