Programming in Python

GAVRILUT DRAGOS

COURSE 3

Sets

Elements from a set can NOT be accessed (they are unordered collections):

Python 2.x / 3.x

```
x = {'A', 'B', 2, 3, 'C'}
x[0], x[1], x[1:2], … ➔ all this expression will produce an error
```

Similarly – there is no addition operation defined between two sets:

Python 2.x / 3.x

```
x = {'A', 'B', 2, 3, 'C'}
y = {'D', 'E', 1}
z = y + z                    #!!!ERROR !!
```

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- Add a new element in the set (either use the member function(method) **add**)

Python 2.x / 3.x

```
x = {1,2,3}           #x = {1, 2, 3}
x.add(4)              #x = {1, 2, 3, 4}
x.add(1)              #x = {1, 2, 3, 4}
```

- Remove a element from the set (methods **remove** or **discard**). Remove throws an error if the set does not contain that element. Use **clear** method to empty an entire set.

Python 2.x / 3.x

```
x = {1,2,3}           #x = {1, 2, 3}
x.remove(1)           #x = {2, 3}
x.discard(2)          #x = {3}
x.discard(2)          #x = {3}
```

```
x = {1,2,3}           #x = {1, 2, 3}
x.clear()             #x = {}
```

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- Union operation can be performed by using the operator | or the method **union**

Python 2.x / 3.x

```
x = {1,2,3}
y = {3,4,5}
t = {2,4,6}
z = x | y | t             #z = {1, 2, 3, 4, 5, 6}
s = {7,8}
w = x.union(s)            #w = {1, 2, 3, 7, 8}
w = x.union(s, y, t)      #w = {1, 2, 3, 4, 5, 6, 7, 8}
```

- **union** method can be called with multiple parameters (sets)

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- Intersection operation can be performed by using the operator & or the method **intersection**

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {2,3,4,5}
t = {3,4,5,6}
z = x & y & t            #z = {3, 4}
w = x.intersection(y)    #w = {2, 3, 4}
w = x.intersection(y, t) #w = {3, 4}
```

- **intersection** method can be called with multiple parameters (sets)

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- Difference operation can be performed by using the operator - or the method **difference**

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {2,3,4,5}
z = x - y              #z = {1}
z = y – x              #z = {5}
w = x.difference(y)    #w = {1}
s = {1,2,3}
w = x.difference(y,s)  #w = {} → empty set
```

- **difference** method can be called with multiple parameters (sets)

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- Symmetric difference operation can be performed by using the operator ^ or the method **symmetric_difference**

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {2,3,4,5}
z = x ^ y                          #z = {1, 5}
z = y ^ x                          #z = {1, 5}
w = x.symmetric_difference(y)      #w = {1, 5}
s = {1,2,3}
w = x.symmetric_difference(y,s)    #!!! ERROR !!!
```

- **symmetric_difference** method can **NOT** be called with multiple parameters (sets)

Sets

Sets support a set of functions that can be used to modify its content. Some of these functionalities can also be achieved by using some operators.

- To test if a an element exists in a set, we can use the **in** operator

Python 2.x / 3.x

```
x = {1,2,3,4}
y = 2 in x                              #y = True
z = 5 not in x                          #z = True
```

- The length of a set can be found out using the **len** keyword

Python 2.x / 3.x

```
x = {10,20,30,40}
y = len (x)                             #y = 4
```

Dictionaries

A dictionary is python implementation of a hash-map container. Design as a (key – value pair) where Key is a unique element within the dictionary.

A special keyword **dict** can be used to create a dictionary. The **{** and **}** can also be used to build a dictionary – much like in the case of sets.

Python 2.x / 3.x

```
x = dict()                          #x is an empty dictionary
x = {}                              #x is an empty dictionary
x = {"A":1, "B":2}                  #x is a dictionary with 2 keys
                                    #("A" and "B")
x = dict(abc=1,aaa=2)               #equivalent to x= {"abc":1,  "aaa":2}
x = dict({"abc":1,"aaa":2})         #equivalent to x= {"abc":1,  "aaa":2}
x = dict([("abc",1) ,("aaa",2)])    #equivalent to x= {"abc":1,  "aaa":2}
x = dict((("abc",1) ,("aaa",2)))    #equivalent to x= {"abc":1,  "aaa":2}
x = dict(zip(["abc","aaa"],[1,2]))#equivalent to x= {"abc":1,  "aaa":2}
```

Dictionaries

Values from a dictionary can also be manipulated with **setdefault** member.

Python 2.x / 3.x

```
x = {"A":1, "B":2}          #x = {"A":1,"B":2}
y = x.setdefault("C",3)     #x = {"A":1,"B":2,"C:3"},              y=3
y = x.setdefault("D")       #x = {"A":1,"B":2,"C:3","D":None},     y=None
y = x.setdefault("A")       #x = {"A":1,"B":2,"C:3","D":None},     y=1
y = x.setdefault("B",20)    #x = {"A":1,"B":2,"C:3","D":None},     y=2
```

Method **update** can also be used to change the value associated with a key.

Python 2.x / 3.x

```
x = {"A":1, "B":2}          #x = {"A":1,"B":2}
x.update({"A":10})          #x = {"A":10,"B":2}
x.update({"A":100,"B":5})   #x = {"A":100,"B":5}
x.update({"C":3})           #x = {"A":100,"B":5,"C":3}
x.update(D=123,E=111)       #x = {"A":100,"B":5,"C":3,"D":123,"E":111}
```

Dictionaries and functional programming

A dictionary can also be built using functional programming

Python 2.x / 3.x

`x = {i:i for i in range(1,9)}`
`#x = {1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8}`

`x = {i:str(64+i) for i in range(1,9)}`
`#x = {1:"A",2:"B",3:"C",4:"D",5:"E",6:"F",7:"G",8:"H"}`

`x = {i%3:i for i in range(1,9)}`
`#x = {0:6,1:7,2:8} → last values that were updated`

`x = {i:str(64+i) for i in range(1,9) if i%2==0}`
`#x = {2:"B", 4:"D", 6:"F", 8:"H"}`

Dictionaries

Operator ** can be used in a function to specify that the list of parameters of that function should be treated as a dictionary.

Python 2.x / 3.x

```
def GetFastestCar(**cars):
       min_speed = 0
       name = None
       for car_name in cars:
              if cars[car_name] > min_speed:
                     name = car_name
                     min_speed = cars[car_name]
       return name
fastest_car = GetFastestCar(Dacia=120,BMW=160,Toyota=140)
print (fastest_car)
#fastes_car = "BMW"
```