Programming in Python

GAVRILUT DRAGOS

COURSE 3

Sets

A list of unique data (two elements **a** and **b** are considered unique if **a** is different than **b** ➔ this translates as **a** is of a different type than **b** or if **a** and **b** are of the same type, that ***a != b***)

A special keyword **set** can be used to create a set. The **{** and **}** can also be used to build a set. Set keyword can be used to initialize a set from tuples ,lists or strings.

Sets supports some special mathematical operations like:

- Intersection
- Union
- Difference
- Symmetric difference

Sets

Python 2.x / 3.x

```
x = set()            #x is an empty set

x = {1,2,3}          #x is a set containing 3 elements: 1,2 and 3
x = {1,2,2,3,1,1}    #x is a set containing 3 elements: 1,2 and 3
x = {1,2,"AB","ab"}  #x is a set containing 4 elements: 1,2,"AB" and "ab"
x = set((1,2,3,2))   #x is a set containing 3 elements: 1,2 and 3
x = set([1,2,3,2])   #x is a set containing 3 elements: 1,2 and 3
x = set("Hello")     #x is a set containing 4 characters: H,e,l and o
```

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.

- Several elements can be added to a set by either use the member function(method) **update** or by using the operator **|=**

Python 2.x / 3.x

```
x = {1,2,3}                 #x = {1, 2, 3}
x |= {3,4,5}                #x = {1, 2, 3, 4, 5}
x.update({5,6})             #x = {1, 2, 3, 4, 5, 6}
x.update({5,6},{6,7})       #x = {1, 2, 3, 4, 5, 6, 7}
x.update({8},{6},{9})       #x = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- **update** 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.

- 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, 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

All sets operations also support some operations that apply to one variable such as:

- Intersection
  - **intersection_update**
  - **&=**
- Difference
  - **difference_update**
  - **-=**
- Symmetric difference
  - **symmetric_difference_update**
  - **^=**

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
```

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.

- Use method **isdisjoint** to test if a set has no common elements with another one

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {10,20,30,40}
z = x.isdisjoint(y)                 #z = True
```

- Use method **issubset** or operator **<=** to test if a set is included in another one

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {1,2,3,4,5,6}
z = x.issubset(y)                   #z = True
t = x <= y                          #t = True
```

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.

- Use method **issuperset** or operator **>=** to test if a set is included in another one

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {1,2,3,4,5,6}
z = y.issuperset(x)                    #z = True
t = y >= x                             #t = True
```

- Operator > can also be used → it checks if a set is included in another **BUT** is not identical to it. Operator < can be used in the same way.

Python 2.x / 3.x

```
x = {1,2,3,4}
y = {1,2,3,4,5,6}
t = y > x                  #t = True
```

```
x = {1,2,3,4}
y = {1,2,3,4}
t = y > x                  #t = False
```

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.

- Use method **pop** to remove one element from the set. The remove element is different from Python 2.x to Python 3.x in terms of the order the element are kept in memory. Even if sets are unordered collection, in order to have quick access to different elements of the set these elements must be kept in memory in a certain way.

Python 2.x / 3.x

```
x = {"A","a","B","b",1,2,3}
print (x)
print (x.pop())
```

Output (Python 3)

```
{1, 2, 3, 'b', 'B', 'A', 'a'}
1
```

Output (Python 2)

```
set(['a', 1, 2, 3, 'b', 'A', 'B'])
a
```

- Use **copy** method to make a shallow copy of a set.

Sets and functional programming

A set can also be built using functional programming

- The main difference is that instead of operator […] to build a set one need to use {…}

Python 2.x / 3.x

```
x = {i for i in range(1,9)}                    #x = {1,2,3,4,5,6,7,8}
x = {i for i in range(1,100) if i % 23 == 0}   #x = {23, 46, 69, 92}
x = {i*i for i in range(1,6)}                  #x = {1, 4, 9, 16, 25}
```

- The condition of the set (all elements are unique) still applies

Python 2.x / 3.x

```
x = {i%5 for i in range(1,100)}                #x = {0, 1, 2, 3, 4|
```

Sets and Built-in functions

The default build-in functions for list can also be used with sets and lambdas.

- Use **map** to create a new set where each element is obtained based on the lambda expression provided.

Python 2.x/3.x

```
x = {1,2,3,4,5}
y = set(map(lambda element: element*element,x))    #y = {1,4,9,16,25}
```

```
x = [1,2,3]
y = [4,5,6]
z = set(map(lambda e1,e2: e1+e2,x,y))              #z = {5,7,9}
```

Sets and Built-in functions

The default build-in functions for list can also be used with sets and lambdas.

- Use **filter** to create a new set where each element is filtered based on the lambda expression provided.

Python 2.x/3.x

```
x = [1,2,3,4,5]
y = set(filter(lambda element: element%2==0,x))       #y = {2,4}
```

- Both **filter** and **map** are used to create a set (usually in conjunction with **range** keyword)

Python 2.x/3.x

```
x = set(map(lambda x: x*x, range(1,10)))
#x = {1, 4, 9, 16, 25, 36, 49, 64, 81}

x = set(filter(lambda x: x%7==1,range(1,100)))
#x = {1, 8, 15, 22, 29, 36, 43, 50, 57, 64, 71, 78, 85, 92, 99}
```

Sets and Built-in functions

The default build-in functions for list can also be used with sets and lambdas.

- Other functions that work in a similar way as the build-in functions for list are: **min**, **max**, **sum**, **any**, **all, sorted, reversed**
- **for** statement can also be used to enumerate between elements of a set

Python 2.x / 3.x

```
for i in {1,2,3,4,5}:
       print(i)
```

- Python language also has another type → *frozenset*. A frozen set has all of the characteristics of a normal set, but it can not be modified. To create a frozen set use the **frozenset** keyword.

Python 2.x/3.x

```
x = frozenset ({1,2,3})
x.add(10)                        #!!!ERROR!!!
```

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

To set a value in a dictionary use [] operator. The same operator can be used to read an existing value. If a value does not exists, an exception will be thrown.

Python 2.x / 3.x

```
x = {}                              #x is an empty dictionary
x["ABC"] = 2                        #x is an dictionary with one key (ABC)
y = x["ABC"]                        #y = 2
y = x["test"]                       #!!! ERROR !!!
```

To check if a key exists in a dictionary, use **in** operator. **len** can also be used to find out how may keys a dictionary has.

Python 2.x / 3.x

```
x = {"A":1, "B":2}                  #x is a dictionary with 2 keys
"A" in x                            #True
len (x)                             #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

To delete an element from a dictionary use **del** keyword or **clear** method

Python 2.x / 3.x

```
x = {"A":1, "B":2}                   #x = {"A":1,"B":2}
del x["A"]                           #x = {"B":2}
x.clear()                            #x is an empty dictionary
del x["C"]                           #!!! ERROR !!! "C" is not a key in x
```

To create a new dictionary you can use **copy** or static method **fromkeys**

Python 2.x / 3.x

```
x = {"A":1, "B":2}                   #x={"A":1,"B":2}
y = x.copy()                         #makes a shallow copy of x
y["C"]=3                             #x={"A":1,"B":2},y={"A":1,"B":2,"C":3}
```

```
x = dict.fromkeys(["A","B"])   #x = {"A":None,"B":None}
x = dict.fromkeys(["A","B"],2) #x = {"A":2,"B":2}
```

Dictionaries

Elements from the dictionary can also be accessed with method **get**

Python 2.x / 3.x

```
x = {"A":1, "B":2}           #x = {"A":1,"B":2}
y = x.get("A")               #y = 1
y = x.get("C")               #y = None
y = x.get("C",123)           #y = 123
```

An element can also be extracted using pop method.

Python 2.x / 3.x

```
x = {"A":1, "B":2}           #x={"A":1,"B":2}
y = x.pop("A")               #x={"B":2}, y = 1
y = x.pop("C",123)           #x={"B":2}, y = 123
y = x.pop("D")               #!!! ERROR !!! Key "D" does not exists
                             #and no default value was provided
```

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

Keys from the dictionary can be obtained with method keys

Python 2.x / 3.x

```
x = {"A":1, "B":2}                  #x = {"A":1,"B":2}
y = x.keys()                        #y = ["A","B"] → an iterable object
```

To iterate all keys from a dictionary:

Python 2.x / 3.x

```
x = {"A":1, "B":2}
for i in x:
        print (i)
```

```
x = {"A":1, "B":2}
for i in x.keys():
        print (i)
```

Output

```
A
B
```

Dictionaries

Values from the dictionary can be obtained with method **values**

Python 2.x / 3.x

```
x = {"A":1, "B":2}                      #x = {"A":1,"B":2}
y = x.values()                          #y = ["1","2"] → an iterable object
```

To iterate all values from a dictionary:

Python 2.x / 3.x

```
x = {"A":1, "B":2}
for i in x.values():
        print (i)
```

Output

```
1
2
```

Output order may be different for different versions of python depending on how data is stored/ordered in memory.

Dictionaries

All pairs from a dictionary can be obtained using the method **items**

Python 2.x / 3.x

```
x = {"A":1, "B":2}                  #x = {"A":1,"B":2}
y = x.items()                       #y = an iterable object (Python 3) or
                                    #a list of tuples for Python 2.
                                    #[ ("A":1) , ("B":2) ]
```

To iterate all keys from a dictionary:

Python 2.x / 3.x

```
x = {"A":1, "B":2}
for i in x.items():
        print (i)
```

Output

```
("A", 1)
("B", 2)
```

Dictionaries

Using the **items** method elements from a dictionary can be sorted according to their value.

Python 2.x / 3.x

```
x = {
       "Dacia"  : 120,
       "BMW"    : 160,
       "Toyota" : 140
    }
for i in sorted(x.items(),key = lambda element : element[1]):
      print (i)
```

Output

```
("Dacia",   120)
("Toyota", 140)
("BMW",   160)
```

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"
```

Dictionaries

Build-in functions such as **filter** can also be used with dictionaries.

Python 2.x / 3.x

```
x = {
       "Dacia"  : 120,
       "BMW"    : 160,
       "Toyota" : 140
    }
y = dict(filter(lambda element : element[1]>=140,x.items()))
#y = {"Toyota":140, "BMW":160}
```

To delete an entire dictionary use **del** keyword.