Tutorial_Session3

September 28, 2018

1 Tuples and Dictionaries

1.1 Tuples

- A tuple is an ordered sequence of objects.
- A tuple may be specified by enclosing in the parentheses, the elements of the tuple (possibly of heterogeneous types), separated by commas.

```
In [1]: myTuple = (4, 6, [2, 8], 'abc', {3, 4},(23,56,12))
```

```
In [2]: digits = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
In [3]: subjects = ('Physics', 'Chemistry', 'Computer Science')
```

```
In [4]: months = ((1, 'January'), (2, 'February'), (3, 'March'))
```

```
In [5]: single=("alpha")
        one=(1)
        print(type(digits),type(subjects),type(months),type(single),type(one))
```

```
<class 'tuple'> <class 'tuple'> <class 'tuple'> <class 'str'> <class 'int'>
```

```
In [6]: one=(1,)
        print(type(one))
```

```
<class 'tuple'>
```

- If a tuple comprises a single element, the element should be followed by a comma to distinguish a tuple from a parenthesized expression.
- A tuple having a single element is also known as singleton tuple.

- Tuples being **immutable**, an attempt to modify an element of a tuple yields an error.

```
In [7]: digits[1] = 3
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-7-93313e5806a7> in <module>()
----> 1 digits[1] = 3


    TypeError: 'tuple' object does not support item assignment
```

```
In [47]: t=((23,5,67),(11,34,98),(9,34,76))
         sum(t)
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-47-804c38f931cb> in <module>()
      1 t=((23,5,67),(11,34,98),(9,34,76))
----> 2 sum(t)


    TypeError: unsupported operand type(s) for +: 'int' and 'tuple'
```

1.2 Tuple Operations

```
In [39]: weekDays = ('Monday', 'Tuesday')
         marks = (78, 99, 34, 45)
         dateOfBirth = (1, 'October', 1990)
```

1.2.1 Multiplication Operator *

```
In [10]: weekDays * 2

Out[10]: ('Monday', 'Tuesday', 'Monday', 'Tuesday')
```

1.2.2 Concatenation Operator +

```
In [40]: print(id(weekDays))
         weekDays = weekDays + ('Wednesday',)
         print(id(weekDays))

1853701287368
1853711261416
```

1.2.3 Length Operator len

```
In [12]: len(weekDays)
```

```
Out[12]: 3
```

1.2.4 Indexing & Slicing

```
In [13]: yearOfBirth = dateOfBirth[-1] #Indexing
         print(yearOfBirth)
```

```
1990
```

```
In [ ]: weekDays[1:2] #Slicing
```

1.2.5 Function min & max

```
In [14]: min(marks)
```

```
Out[14]: 34
```

```
In [15]: max(marks)
```

```
Out[15]: 99
```

1.2.6 Function sum

```
In [16]: sum(marks)
```

```
Out[16]: 256
```

1.2.7 Membership Operator in

```
In [17]: 'Friday' in weekDays
```

```
Out[17]: False
```

1.2.8 Function tuple

- The function tuple can be used to convert a sequence to a tuple.

```
In [18]: vowels = 'aeiou'
         T=tuple(vowels)
         l=[12,45,67]
         T1=tuple(l)
         print(T,T1)
```

```
('a', 'e', 'i', 'o', 'u') (12, 45, 67)
```

1.3 Unpacking operation

```
In [25]: t=(23,45,67,(34,21))
         t1,t2,t3,t4=t,t,t,t
         print( t1,t2,t3,t4)
         print(type(t4))
```

```
(23, 45, 67, (34, 21)) (23, 45, 67, (34, 21)) (23, 45, 67, (34, 21)) (23, 45, 67, (34, 21))
<class 'tuple'>
```

1.4 Built-in Functions on Tuples

1.4.1 Function count

- Returns count of occurrences of an element in the tuple.

```
In [27]: age = (20, 18, 17, 19, 18, 18)
         print(age.count(18))
```

```
3
```

1.4.2 Function index

- Returns index of the first occurrence of an element in the tuple.

```
In [28]: age.index(18)
```

```
Out[28]: 1
```

1.5 Problem: Sort list of tuples

```
In [29]: def sortList(studentList):
             '''
             Objective: To sort a given list of tuples in increasing order on the basis
                        of marks.
             Input Parameter: studentList: a list of tuples such that each tuple
                              contains student names and their marks
             Return Value: studentList: sorted list
             '''
             '''
             Approach:
                 Use bubble sort.
                 Starting from the first element (tuple), compare two consecutive elements
                 of the list and swap the two if marks in second element is smaller than the
                 marks in first element. At the end of first iteration, smallest element in
                 the list will come in the beginning. If there are n elements, the above
                 procedure will repeat n-1 times
             '''
```

```
    for i in range(0, len(studentList) - 1):
        for j in range(0, len(studentList) - 1 - i):
            if studentList[j+1][1] < studentList[j][1]:
                studentList[j+1], studentList[j] = studentList[j], studentList[j+1]
    return studentList

def main():
    '''
    Objective: To call a function sortList to sort a given list of tuples in
    increasing order on the basis of marks provided as input.
    Input Parameter: None
    Return Value: None
    '''
    #studentList = [('Rohit', 50), ('Deepak', 75), ('Sonali', 47)]
    studentList = []
    num = int(input('Enter the number of students:'))
    for i in range(num):
        pair = eval(input('Enter tuple <student name, marks>'))
        studentList.append(pair)
    sortList(studentList)
    print(studentList)

if __name__ == '__main__':
    main()
```

```
Enter the number of students:5
Enter tuple <student name, marks>('raman',67)
Enter tuple <student name, marks>('Reeta',98)
Enter tuple <student name, marks>('Sarita',75)
Enter tuple <student name, marks>('Neha',82)
Enter tuple <student name, marks>('Gurnam',88)
[('raman', 67), ('Sarita', 75), ('Neha', 82), ('Gurnam', 88), ('Reeta', 98)]
```

1.6 Dictionaries

- A dictionary is an unordered sequence of key-value pairs.
- Key and value in a key-value pair in a dictionary are separated by a colon. Further, the key:value pairs in a dictionary are separated by commas and are enclosed between curly parentheses.
- Indices in a dictionary can be of any immutable type and are called keys.

```
In [50]: month = {}
         month[1] = 'Jan'
         month[2] = 'Feb'
         month[3] = 'Mar'
         month[4] = 'Apr'
         print(month)
         print(month[8])
```

```
{1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr'}
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-50-6d24bc427f54> in <module>()
          5 month[4] = 'Apr'
          6 print(month)
    ----> 7 print(month[8])


        KeyError: 8
```

- The search in a dictionary is based on the key.
- Therefore, in a dictionary, the keys are required to be unique. However, the same value may be associated with multiple keys.

```
In [52]: price = {'tomato':40, 'cucumber':30,
         'potato':20, 'cauliflower':70, 'cabbage':50,
         'lettuce':40, 'raddish':30, 'carrot':20,
         'peas':80}
```

- Values associated with keys can be mutable objects and thus, may be changed at will.

```
In [53]: price['tomato'] = 25
         print(price)
```

```
{'tomato': 25, 'cucumber': 30, 'potato': 20, 'cauliflower': 70, 'cabbage': 50, 'lettuce': 40,
```

- Keys in a dictionary may be of heterogeneous types

```
In [ ]: counting = {1:'one', 'one':1, 2:'two', 'two':2}
```

1.7 Dictionary Operations

```
In [54]: digits = {0:'Zero', 1:'One', 2:'Two', 3:'Three', 4:'Four', 5:'Five', 6:'Six', 7:'Seve
```

1.7.1 length Operator len

```
In [55]: len(digits)
```

```
Out[55]: 10
```

1.7.2 Indexing

```
In [56]: digits[1]
```

```
Out[56]: 'One'
```

1.7.3 Functions min and max

```
In [57]: min(digits)
```

```
Out[57]: 0
```

```
In [58]: max(digits)
```

```
Out[58]: 9
```

1.7.4 Function sum

```
In [59]: sum(digits)
```

```
Out[59]: 45
```

1.7.5 Membership Opeartor in

```
In [60]: 5 in digits
```

```
Out[60]: True
```

```
In [61]: 'Five' in digits
```

```
Out[61]: False
```

Note: Membership operation in, and functions min, max and sum apply only to the keys in a dictionary.

1.7.6 Deleting a key-value pair from dictionary

```
In [62]: winter = {11:'November ', 12: 'December', 1:'January', 2:'February'}
         del winter[11]
         print(winter)
```

```
{12: 'December', 1: 'January', 2: 'February'}
```

1.8 Built-in Functions on Dictionaries

1.8.1 Deleting all key-value pairs using clear function

```
In [ ]: winter.clear()
        print(winter)
```

1.8.2 Function get

- The function get is used to extract the value corresponding to a given key
- The first parameter is used to specify the key and the second parameter is used to specify the value to be returned in case the key is not found in the dictionary. In case, the second parameter is not specified, the system returns None

```
In [63]: passwords = {'Ram':'ak@607', 'Shyam':'rou.589'}

In [64]: passwords.get('Ram',-1)

Out[64]: 'ak@607'

In [65]: passwords.get('Raman',-1)

Out[65]: -1
```

1.8.3 Function update

- The function update is used to insert in a dictionary, all the key–value pairs of another dictionary

```
In [68]: morePasswords = {'Ram': 'ytu7','Raman':'vi97@4', 'Kishore':'23@0jsk'}

In [69]: passwords.update(morePasswords)
         print(passwords)

{'Ram': 'ytu7', 'Shyam': 'rou.589', 'Raman': 'vi97@4', 'Kishore': '23@0jsk'}
```

1.8.4 Function keys

- Return an object comprising of all keys of the dictionary.

```
In [70]: months = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr'}
         print(months.keys())

dict_keys([1, 2, 3, 4])
```

1.8.5 Function values

- Return an object comprising of all values of the dictionary.

```
In [71]: print(months.values())

dict_values(['Jan', 'Feb', 'Mar', 'Apr'])
```

1.8.6 Function items

- Return an object comprising of tuples of key-value pairs present in the dictionary.

```
In [72]: print(months.items())
```

```
dict_items([(1, 'Jan'), (2, 'Feb'), (3, 'Mar'), (4, 'Apr')])
```

1.9 Dictionary of state and its capitals

```
In [73]: def stateCapitalDict():
             '''
             Purpose: To form a dictionary of state and its capital as specified by user.
             Input Parameter: None
             Return Value: stateCapital - Dictionary containing state as keys and capital
                           as values
             '''
             '''
             Approach:
             For each state and capital taken as input from the user
                 Assign value capital to the key state
             '''
             stateCapital = dict()
             state = input('Enter state:')
             capital = input('Enter capital:')
             while state != '':
                 stateCapital[state] = capital
                 state = input('Enter state:')
                 capital = input('Enter capital:')
             return stateCapital

         def main():
             '''
             Purpose: To form a dictionary of state and its capital as specified by user.
             Input Parameter: None
             Return Value: None
             '''
             dict1 = stateCapitalDict()
             print(dict1)

         if __name__ == '__main__':
             main()
```

```
Enter state:Delhi
Enter capital:Delhi
Enter state:UP
Enter capital:Lucknow
Enter state:West Bengal
Enter capital:Kolkata
Enter state:
Enter capital:
{'Delhi': 'Delhi', 'UP': 'Lucknow', 'West Bengal': 'Kolkata'}
```

```
In [74]: message='ZghTe KIAeo GlyUa Ehipd Inosea'
         D={}
         print(message)
         for ch in message:
             if ch in D:
                 D[ch]+=1
             else:
                 D[ch]=1
         print(D)
         print("message lengh=",len(message))
         print("Sum of counts=",sum(D.values()))
```

```
ZghTe KIAeo GlyUa Ehipd Inosea
{'Z': 1, 'g': 1, 'h': 2, 'T': 1, 'e': 3, ' ': 4, 'K': 1, 'I': 2, 'A': 1, 'o': 2, 'G': 1, 'l':
message lengh= 30
Sum of counts= 30
```