Lecture07

February 20, 2024

[1]:
```
# Data Structures
# list  +, *, in, len, append, count, index, .., slicing, indexing
# tuple
# set
# dict
```

1 List

[13]:
```
L=[1,2,4,5,10, True, 4.54]
L[0]  # indexing
L[-1] # indexing
# L[start:end:step]   # slicing
L[:4]      # slicing
L[::2]     # slicing
L[5:1:-1] # slicing
L[::-1]   # slicing
```

[13]:
```
[4.54, True, 10, 5, 4, 2, 1]
```

1.0.1 Shallow vs Deep Copy

[18]:
```
# Shallow copy for L1
L1=[1,2,3]
L2=[4,5, L1]
L1[0]=100
print(L1)
print(L2)
```

```
[100, 2, 3]
[4, 5, [100, 2, 3]]
```

[19]:
```
# Deep Copy for L1
L1=[1,2,3]
L2=[4,5, list(L1)]
L1[0]=100
print(L1)
print(L2)
```

```
[100, 2, 3]
[4, 5, [1, 2, 3]]
```

[16]:
```
# Shallow copy for L1 and L2
L1=[1,2,3]
L2=[4,5, L1]
L3=[6,7,L2]  # L3  --> [6,7, ref_to_L2 ]
L1[0]=100
print(L1)
print(L2)
print(L3)
```

```
[100, 2, 3]
[4, 5, [100, 2, 3]]
[6, 7, [4, 5, [100, 2, 3]]]
```

[20]:
```
# Deep Copy for L2 but shallow copy for L1
L1=[1,2,3]
L2=[4,5, L1]
L3=[6,7,list(L2)]   # L3  --> [6,7, [4,5, ref_to_L1]]
L1[0]=100
print(L1)
print(L2)
print(L3)
```

```
[100, 2, 3]
[4, 5, [100, 2, 3]]
[6, 7, [4, 5, [100, 2, 3]]]
```

[23]:
```
# Deep copy
import copy
L1=[1,2,3]
L2=[4,5, L1]
L3=[6,7,copy.deepcopy(L2)]   # L3  --> [6,7, [4,5, [1,2,3]]]
L1[0]=100
print(L1)
print(L2)
print(L3)
```

```
[100, 2, 3]
[4, 5, [100, 2, 3]]
[6, 7, [4, 5, [1, 2, 3]]]
```

[24]:
```
# Deep copy
L1=[1,2,3]
L2=[4,5, list(L1)]
```

```
L3=[6,7,list(L2)]    # L3  --> [6,7, [4,5, [1,2,3]]]
L1[0]=100
print(L1)
print(L2)
print(L3)
```

```
[100, 2, 3]
[4, 5, [1, 2, 3]]
[6, 7, [4, 5, [1, 2, 3]]]
```

2 tuple

[42]:
```
# is similar to list but (immutable)
```

[29]:
```
#  create tuple
t=tuple()
t=()
t=(1,3,4,5,6)
```

[32]:
```
# tuple can contain any type of items
t=(1,2,3,True, 4.5, "welcome")
t
```

[32]:
```
(1, 2, 3, True, 4.5, 'welcome')
```

[36]:
```
# Access items (indexing)
t[-1]
t[3]
```

[36]:
```
True
```

[38]:
```
# Slicing
# t[start: end: step]
t[1:5]
t[:5:2]
```

[38]:
```
(1, 3, 4.5)
```

[41]:
```
# Operators
# +, *, in, len, count, index
t1=(1,2,3)
t2=(4,5, False)
t1+t2
t1*4
4 in t1
```

```
[41]: False
```

2.0.1 Mutablility feature in tuples

```
[46]: # lists are mutable
      print(L)
      L[0]=100
      print(L)
```

```
[100, 2, 4, 5, 10, True, 4.54]
[100, 2, 4, 5, 10, True, 4.54]
```

```
[49]: # tuples are immutable
      print(t)
      # t[0]=100    # NOT POSSIBLE
      # print(t)
```

```
(1, 2, 3, True, 4.5, 'welcome')
```

2.0.2 Nested tuples

```
[50]: t1=(1,2,3)
      t2=(4,5,"Welcome")
      t3=(t1, "thank you", t2)
```

```
[52]: print(t3)
      print(len(t3))
```

```
((1, 2, 3), 'thank you', (4, 5, 'Welcome'))
3
```

3 set

```
[58]: # create sets
      s=set()
      s={1,2,4,5}    # this is ok to create set
      # s={}           # this is NOT ok to create set (this will create dict)
      type(s)
```

```
[58]: set
```

```
[60]: # set stores one instance of the value only (it doesn't store duplicates)
      s={1,2,3,1,1,1,1, "hasan", 'james', "hasan", 1, "hasan"}
      print(s)
```

```
{1, 2, 'james', 3, 'hasan'}
```

```
[64]: # operators that we can use with set
      # in, &, |, -, ^
      print('hasan' in s)
```

```
True
```

```
[65]: s1={'hasan', 'alma', 'sara', 'mike'}
      s2={'sara', 'james', 'alma'}
```

```
[69]: # and (what is shared between the two sets)
      s1 & s2
      s1.intersection(s2)
      s2.intersection(s1)
```

```
[69]: {'alma', 'sara'}
```

```
[72]: # or (what is contained in both s1 and s2 , i.e. union)
      s1 | s2
      s1.union(s2)
      s2.union(s1)
```

```
[72]: {'alma', 'hasan', 'james', 'mike', 'sara'}
```

```
[74]: # difference (what is contained in s1 NOT in s2)
      s1-s2
      s1.difference(s2)
```

```
[74]: {'hasan', 'mike'}
```

```
[76]: s2-s1
      s2.difference(s1)
```

```
[76]: {'james'}
```

```
[80]: # symmetric difference (what is NOT shared between s1 and s2, i.e.␣
        ↪(s1-s2)|(s2-s1) )
      s1^s2
      s2^s1
      s1.symmetric_difference(s2)
      s2.symmetric_difference(s1)
```

```
[80]: {'hasan', 'james', 'mike'}
```

4 dict

[87]:
```
# dictionarys are mutable
```

[88]:
```
# create dictionary
d=dict()
d={}
d
```

[88]:
```
{}
```

[117]:
```
salaries={'james':2000, 'sara':3000,'maya':5000}
```

[118]:
```
salaries['sara']=6000
```

[119]:
```
print(salaries.values())
print(salaries.keys())
print(salaries.items())
```

```
dict_values([2000, 6000, 5000])
dict_keys(['james', 'sara', 'maya'])
dict_items([('james', 2000), ('sara', 6000), ('maya', 5000)])
```

[120]:
```
list(salaries.values())[0]
list(salaries.keys())[0]
list(salaries.items())[0]
```

[120]:
```
('james', 2000)
```

[122]:
```
# dictionary's values can be anything
salaries['maya'] = 'Five Thousands'
```

[123]:
```
salaries['maya']=[200,400, 1500, 700]
```

[125]:
```
salaries
```

[125]:
```
{'james': 2000, 'sara': 6000, 'maya': [200, 400, 1500, 700]}
```

[126]:
```
salaries['james'] = {'jan': 200, 'may':500}
```

[127]:
```
salaries
```

[127]:
```
{'james': {'jan': 200, 'may': 500},
 'sara': 6000,
 'maya': [200, 400, 1500, 700]}
```

```
[130]: # dictionay's keys can be any immutable thing
       salaries['hasan']=5000
```

```
[132]: salaries[('mike','robert')]= 6000
```

```
[135]: salaries[112] = 7000
```

```
[137]: salaries[('tara',1995)] = 5500
```

```
[138]: salaries
```

```
[138]: {'james': {'jan': 200, 'may': 500},
        'sara': 6000,
        'maya': [200, 400, 1500, 700],
        'hasan': 5000,
        ('mike', 'robert'): 6000,
        112: 7000,
        ('tara', 1995): 5500}
```

```
[141]: # salaries[['mike','smith']]=2000    # Does't work
```

Operators, functions and methods

```
[145]: # in   (is used to verify if exist in keys)
       # 5500 in salaries          # 5500 in salaries.keys()
       # 'hasan' in salaries       #'hasan' in salaries.keys()
```

```
[146]: # del
       del salaries['hasan']
```

5 Convert between data structures

```
[158]: # list(), tuple(), set(), dict()
       L=[1,2,3]
       tuple(L)
```

```
[158]: (1, 2, 3)
```

```
[162]: t=(10,20,30)
       list(t)
```

```
[162]: [10, 20, 30]
```

```
[166]: L=[1,2,3,2,3,3,3]
       set(L)
```

[166]:
```
{1, 2, 3}
```

[169]:
```
s={1,2,3}
list(s)
list(t)
```

[169]:
```
[10, 20, 30]
```

[171]:
```
L=[1,2,3]
# dict(L)    # Not possible
```

[157]:
```
# convert list of tuples into dict
LL= [('hasan',40), ('sara',20), ('william', [10,20,30])]
d=dict(LL)
d
```

[157]:
```
{'hasan': 40, 'sara': 20, 'william': [10, 20, 30]}
```

[155]:
```
# convert dict into list of tuples
list(d.items())
```

[155]:
```
[('hasan', 40), ('sara', 20), ('william', [10, 20, 30])]
```

[179]:
```
print(list(d))
print(list(d.keys()))
print(list(d.values()))
print(list(d.items()))
```

```
['hasan', 'sara', 'william']
['hasan', 'sara', 'william']
[40, 20, [10, 20, 30]]
[('hasan', 40), ('sara', 20), ('william', [10, 20, 30])]
```

[180]:
```
print(tuple(d))
print(tuple(d.keys()))
print(tuple(d.values()))
print(tuple(d.items()))
```

```
('hasan', 'sara', 'william')
('hasan', 'sara', 'william')
(40, 20, [10, 20, 30])
(('hasan', 40), ('sara', 20), ('william', [10, 20, 30]))
```

[183]:
```
print(set(d))
print(set(d.keys()))
# print(set(d.values()))
# print(set(d.items()))
```

```
{'william', 'sara', 'hasan'}
{'william', 'sara', 'hasan'}
```

5.1 Some important functions

[187]:
```
print(L)
print(t)
print(s)
print(d)
```

```
[1, 2, 3]
(10, 20, 30)
{1, 2, 3}
{'hasan': 40, 'sara': 20, 'william': [10, 20, 30]}
```

[196]:
```
# len, sum, max, min, zip
print(len(L))
print(len(t))
print(len(s))
print(len(d))    # len(d.keys())
```

```
3
3
3
3
```

[195]:
```
x=[1,2,3, 'welcome', True, [2,4,5]]
sum(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[195], line 2
      1 x=[1,2,3, 'welcome', True, [2,4,5]]
----> 2 sum(x)

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

[202]:
```
#  using the function zip
L1=[1,2,3]
L2=['james','sara','alma','william']
L3=[2000,3000,5000,2500]
L=list(zip(L1,L2, L3))
```

[203]:
```
L
```

[203]:
```
[(1, 'james', 2000), (2, 'sara', 3000), (3, 'alma', 5000)]
```

```
[209]: # unzip using the function zip
       list(zip(*L))
       list(list(zip(*L))[0])

[209]: [1, 2, 3]

[216]: y= list(zip(*L))
       y

[216]: [(1, 2, 3), ('james', 'sara', 'alma'), (2000, 3000, 5000)]

[211]: x1,x2,x3= list(zip(*L))

[212]: x1

[212]: (1, 2, 3)

[213]: x2

[213]: ('james', 'sara', 'alma')

[214]: x3

[214]: (2000, 3000, 5000)

[221]: L=[1,2,3]
       dict(list(zip(L,L)))

[221]: {1: 1, 2: 2, 3: 3}

[ ]:
```