PythonCourse_9_TuplesDictionary&Sets

April 18, 2021

0.1 Tuples

[2]:

```
#Tuples are similar to lists

a = (1,2)
print('a', a, type(a))
```

```
a (1, 2) <class 'tuple'>
```

[5]:

```
#When many values are put together Python by default assumes them as tuples
b = 1,2
print('b', b, type(b))

c,d = 1,2 #not a tuple
print('c', c, type(c))

e = 1,'a'
print('e', e, type(e))
```

```
b (1, 2) <class 'tuple'>
c 1 <class 'int'>
e (1, 'a') <class 'tuple'>
```

[7]:

```
## Accessing, Indexing, Slicing

# Similar to lists
a = (1,2,3,4,5,6)
print(a[0])
print(a[-1])
print(a[3:])
print(a[1:5:2])
#print(a[9]) #IndexError: tuple index out of range
```

```
1
6
(4, 5, 6)
(2, 4)
```

[13]:

```python
## Tuples are Immutable

# This is the difference between lists and tuples
a = (3,4,5,6)
print(a[0])
#a[0] = 5 # TypeError: 'tuple' object does not support item assignment

# changing, appending, deleting and element from tuple not possible
#del a[2] #TypeError: 'tuple' object doesn't support item deletion

# Deletion of entire tuple possible
del a
#print(a) #NameError: name 'a' is not defined
```

```
3
```

[28]:

```python
## Functions used with Tuples

a = (1,2,3)

# for loops
for element in a:
    print(element)

# membership
print(1 in a)
print(9 in a)
print(9 not in a)

# length
print('length',len(a))

# list to tuple
li = [1,2,3]
print(tuple(li)) #creates a new tuple
print(type(li)) #li type does not change
```

```
1
2
3
True
False
True
length 3
(1, 2, 3)
<class 'list'>
```

[23]:

```
## Functions used with Tuples - cont

a = (34,32,7,40,11)

# min, max
print(min(a))
print(max(a))

b = (2,(1,2),'a')
#print(min(b)) # TypeError: '<' not supported between instances of 'tuple' and 'int'
# must be of comparable types
c = (2, 2.2,3.4) #eg: int and float are comparable
print(max(c))
```

```
7
40
3.4
```

[26]:

```
## Functions used with Tuples - cont

a = (1,2,3)
b = 4,5,6

# concatenation
c = a + b
print(c)

# tuple of tuples
d = (a,b)
print(d)

# repetition
e = a * 3
print(e)
```

```
(1, 2, 3, 4, 5, 6)
((1, 2, 3), (4, 5, 6))
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

[8]:

```
## Variable Input to Functions

def sum(a,b,*c): #*variable creates variable length input
    print('a', a, type(a))
    print('b', b, type(b))
    print('c', c, type(c)) #variable length input created is a tuple

    sum = a + b
    for i in c: #iterating through the tuple
        sum += i
    return sum

a,b,c,d = 1,2,3,4
ans1 = sum(a,b) #only two args passed, but a variable length input created␣
 ↪becaude of the func def
ans2 = sum(a,b,c,d) #when more args given, it goes inside the tuple
print('ans1', ans1)
print('ans2', ans2)
```

```
a 1 <class 'int'>
b 2 <class 'int'>
c () <class 'tuple'>
a 1 <class 'int'>
b 2 <class 'int'>
c (3, 4) <class 'tuple'>
ans1 3
ans2 10
```

[15]:

```
## Variable Output from Functions

def sum_diff(a ,b):
    return a + b, a - b #when returning more than 1 variable, Python by default␣
 ↪creates a tuple

ans = sum_diff(5, 4) #when taken into a single variable, we get a tuple
print('ans', ans, type(ans))
sum,diff = sum_diff(5, 4) #when taken into separate variables, they are int
print('sum', sum, type(sum))
print('diff', diff, type(diff))

#Note: unpacking should be right, we cannot assign three values returned to two␣
 ↪variable
def sum_diff_mul(a ,b):
    return a + b, a - b, a * b

#sum,diff = sum_diff_mul(5, 4) #ValueError: too many values to unpack (expected␣
 ↪2)
sum,diff,mul = sum_diff_mul(5, 4)
print(sum,diff,mul)
print(sum_diff_mul(5, 4))
```

```
ans (9, 1) <class 'tuple'>
sum 9 <class 'int'>
diff 1 <class 'int'>
9 1 20
(9, 1, 20)
```

0.2 Dictionary

[19]:

```
## Creating a dictionary

a = {}
print(a,type(a))

d = {"the":2, "a":5, 10000:"str"}
print(d)
print('Length of d is', len(d))
# Note: For Index 10000 to be cretaed, we needed a list of length 10000 + 1
#       But using dictionaries it can be easily created ; Here length of dict␣
 ↪is still 3
```

```
{} <class 'dict'>
{'the': 2, 'a': 5, 10000: 'str'}
Length of d is 3
```

[23]:

```
## Some other ways to create a Dictionary

#Copy an existing dictionary
b = d.copy()
print('b', b)

#Using a List of Tuples
c = dict([("the",2),("a",5),(600,"Hi")])
print('c', c)

#Using fromkeys keyword
e = dict.fromkeys(["hello", 2, 5.5]) #Pass the keys in a list
print('e', e) #Values are none by default
f = dict.fromkeys(["hello", 2, 5.5], 10) #Second arg gives the value ; assigned␣
 ↪to all the keys
print('f', f)
```

```
b {'the': 2, 'a': 5, 10000: 'str'}
c {'the': 2, 'a': 5, 600: 'Hi'}
e {'hello': None, 2: None, 5.5: None}
f {'hello': 10, 2: 10, 5.5: 10}
```

[30]:

```
## Accessing elements in a Dictionary

d = {1:2, 3:4, "list":[1,23], "dict":{5:6}}
print('d',d)

#print(d[0])# KeyError: 0 ; There is no key as 0
print('d[1]',d[1])
print('d["list"]',d["list"])
```

```
#print('d["li"]',d["li"]) # KeyError: 'li' ; No such key
```

```
d {1: 2, 3: 4, 'list': [1, 23], 'dict': {5: 6}}
d[1] 2
d["list"] [1, 23]
```

[37]:

```
## Accessing elements in a Dictionary - Another Mothod

print(d.get(1)) #key present ; Returns the value
print(d.get(0)) #key not present ; Returns None; No Error

print(d.get(0,"Not There")) #key not present ; Returns the second arg
print(d.get(1,"Not There")) #key present ; Returns the value
```

```
2
None
Not There
2
```

[41]:

```
## Some other Methods used

print(d.keys()) # Returns all the keys as a list
print(d.values()) # Returns all the values as a list
print(d.items()) # Returns the key- value pairs as tuples within list
```

```
dict_keys([1, 3, 'list', 'dict'])
dict_values([2, 4, [1, 23], {5: 6}])
dict_items([(1, 2), (3, 4), ('list', [1, 23]), ('dict', {5: 6})])
```

[50]:

```
## Looping through Dictionary

for i in d: #loops through the keys
    print('key', i, 'value', d[i])

print()
for i in d.values(): #loops through the values
    print('value', i)
```

```
key 1 value 2
key 3 value 4
key list value [1, 23]
key dict value {5: 6}

value 2
value 4
value [1, 23]
value {5: 6}
```

[53]:

```
## Membership in Dictionary

print("list" in d) # determines if the KEY exists in the Dict or not
print("li" in d)
print(2 in d) # 2 is a value here ; Therefore returns false
```

```
True
False
False
```

[4]:

```
## Adding Elements to a Dictionary

d = {1:2, 3:4, "list":[1,23], "dict":{5:6}}
print(d)
d['tuple']= (7,9) #Adds the new key and value
print('After adding',d)

# Update data
d[1] = 10 #dict[key] = new value
print('After updating',d)
```

```
{1: 2, 3: 4, 'list': [1, 23], 'dict': {5: 6}}
After adding {1: 2, 3: 4, 'list': [1, 23], 'dict': {5: 6}, 'tuple': (7, 9)}
After updating {1: 10, 3: 4, 'list': [1, 23], 'dict': {5: 6}, 'tuple': (7, 9)}
```

[16]:

```
## Update Function

a = {1:2, 3:4, "list":[1,23], "dict":{5:6}}
b = {3:5, 2:100, "the":56} #say another dict has some common keys and some new␣
 ↪keys
print('a', a)
print('b', b)


a.update(b) # if key is common--> updates the values of a as like b, if key is␣
 ↪not present--> adds the key to a from b
print('After update function')
print('a', a)
print('b', b)
```

```
a {1: 2, 3: 4, 'list': [1, 23], 'dict': {5: 6}}
b {3: 5, 2: 100, 'the': 56}
After update function
a {1: 2, 3: 5, 'list': [1, 23], 'dict': {5: 6}, 2: 100, 'the': 56}
b {3: 5, 2: 100, 'the': 56}
```

[17]:

```
## Removing elements from a dictionary
```

```
print(a.pop('the')) # key should be given as arg; returns the value of that key␣
 ↪and removes that key from dict
print(a)

# Aliter
del a[1]
print(a)
```

```
56
{1: 2, 3: 5, 'list': [1, 23], 'dict': {5: 6}, 2: 100}
{3: 5, 'list': [1, 23], 'dict': {5: 6}, 2: 100}
```

[18]:

```
## Clearing entire dictionary

a.clear() # dict still present, but empty
print(a)

## Deleting entire dictionary
del a #deletes the entire dict, dict no longer present
#print(a) #ameError: name 'a' is not defined
```

```
{}
```

[26]:

```
## Print Words with Frequency k

def printWordsOfFreqK(string, k):
    word_list = string.split() # get the word list
    #print(word_list)
    d = {} # create a dict
    for word in word_list:
        #if word in d: # if word already present in dict --> increment value
            #d[word] = d[word] + 1
        #else: #if word not present add with value one
            #[word] = 1
        # aliter---Simplified code -->  using get function
        d[word] = d.get(word, 0) + 1
    for key in d:
        if d[key] == k:
            print(key)

string = "She sells sea shells on the sea shore and she sold many many sea␣
 ↪shells"
k = 2
printWordsOfFreqK(string, k)
```

```
shells
many
```

0.3 Sets

[80]:

```
a = {} # By default it is an empty dictionary
print(a, type(a))

a = set() # To create an empty set
print(a, type(a))

a = {1, "abc", 59, "hello"} # sets is a collection of data
print(a, type(a))

a = {1, "abc", 59, "hello", "hello", "hello"} # it is a collection of UNIQUE␣
 ↪data
print(a, type(a))

#Sets do not have any ordering or indexing
#print(a[0]) # TypeError: 'set' object is not subscriptable

print('length', len(a))
```

```
{} <class 'dict'>
set() <class 'set'>
{'hello', 1, 59, 'abc'} <class 'set'>
{'hello', 1, 59, 'abc'} <class 'set'>
length 4
```

[47]:

```
## Membership in Sets

print('abc' in a)
print('ijk' in a)
```

```
True
False
```

[53]:

```
## Loopin through Sets
for element in a:
    print(element) # prints RANDOMLY !
```

```
hello
1
59
abc
```

[64]:

```
## Adding elements in Sets

a.add('Jay')
print(a)

# update function
```

```
b = {'mno', 'Jay', 88}
a.update(b) # adds element from b to a if not already present in a
print(a)
```

```
{1, 'Jay', 'hello', 59, 'abc'}
{1, 'mno', 'Jay', 'hello', 88, 59, 'abc'}
```

[65]:

```
## Removing Elements from Sets

a.remove('mno')
print(a)
#a.remove('zzz') # KeyError: 'zzz' ; Create error if element not present

a.discard('hello')
print(a)
a.discard('zzz') # Does not create any error if element not present
print(a)
```

```
{1, 'Jay', 'hello', 88, 59, 'abc'}
{1, 'Jay', 88, 59, 'abc'}
{1, 'Jay', 88, 59, 'abc'}
```

[66]:

```
## other methods for removing

a.pop() #removes RANDOMLY one element
print(a)

a. clear() #clears the set
print(a)

del a #deletes the set
#print(a) NameError: name 'a' is not defined
```

```
{'Jay', 88, 59, 'abc'}
set()
```

[69]:

```
## Functions in Sets

a = {1,2,3,4}
b = {3,4,5,6}

print('A intersection B', a.intersection(b))
print('B intersection A', b.intersection(a))

print('A union B', a.union(b))
print('B union A', b.union(a))

print('A difference B', a.difference(b)) # In A not in B
print('B difference A', b.difference(a)) # In B not in A
```

```python

print('A symmetric difference B', a.symmetric_difference(b)) # (A Union B) - (A Intersection B)
print('B symmetric difference A', b.symmetric_difference(a)) # (B Union A) - (B Intersection A)
```

```
A intersection B {3, 4}
B intersection A {3, 4}
A union B {1, 2, 3, 4, 5, 6}
B union A {1, 2, 3, 4, 5, 6}
A difference B {1, 2}
B difference A {5, 6}
A symmetric difference B {1, 2, 5, 6}
B symmetric difference A {1, 2, 5, 6}
```

[75]:

```python
## Some more Functions in Sets

# These update the original set
a = {1,2,3,4}
b = {3,4,5,6}
print('A intersection update B', a.intersection_update(b)) # Does not return any value ie,returns None
print(a) # A gets updated

a = {1,2,3,4}
b = {3,4,5,6}
#print('A union update B', a.union_update(b)) # AttributeError: 'set' object has no attribute 'union_update'; No Union Update

print('A difference update B', a.difference_update(b)) # Does not return any value ie,returns None
print(a) # A gets updated

a = {1,2,3,4}
b = {3,4,5,6}
print('A symmetric difference update B', a.symmetric_difference_update(b)) # Does not return any value ie,returns None
print(a) # A gets updated
```

```
A intersection update B None
{3, 4}
A difference update B None
{1, 2}
A symmetric difference update B None
{1, 2, 5, 6}
```

[79]:

```
## Some More Functions in Sets

a = {1,2,3,4}
b = {3,4,5,6}
c = {1,2}
d = {6,7,8,9}

print(c.issubset(a))
print(a.issubset(c))

print(c.issuperset(a))
print(a.issuperset(c))

print(a.isdisjoint(b))
print(a.isdisjoint(d))
```

```
True
False
False
True
False
True
```

[81]:

```
## Sum of Unique Nos in a list

def sumUnique(l):
    s = set()
    for i in l:
        s.add(i)
    sum = 0
    for i in s:
        sum += i
    return sum

ans = sumUnique([1,1,2,1,3,4,2,5,4,5,5,2,1])
print(ans)
```

```
15
```