Tuples and Sets

January 25, 2023

1 Tuples

- Tuple is an ordered collection of elements
- Unlike lists tuple are immutable
- Values inside a tuple cannot be changed once created

[3]:
```
lst = [10, 20, 30]
lst[1] = 50
print(lst)
```

```
[10, 50, 30]
```

[5]:
```
t = (10, 20, 30)
print(t)
print(t[:2])
t[1] = 50
print(t)
```

```
(10, 20, 30)
(10, 20)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [5], in <cell line: 4>()
      2 print(t)
      3 print(t[:2])
----> 4 t[1] = 50
      5 print(t)

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

[7]:
```
lst = [10, 20, 30] # lists are mutable
lst.append(40)
print(lst)
lst.reverse()
print(lst)
```

```
[10, 20, 30, 40]
[40, 30, 20, 10]
```

[8]:
```
t = (10, 20, 30)
print(t.count(10))
```

```
1
```

[10]:
```
t = (10, 20, 30)
print(t.index(100))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [10], in <cell line: 2>()
      1 t = (10, 20, 30)
----> 2 print(t.index(100))

ValueError: tuple.index(x): x not in tuple
```

2 Sets

- Set is an unordered collection of unique elements (immutable)
- Sets are mutable
- Set elements are not subscriptable
- Sets will not hold duplicate values

[11]:
```
lst = [10, 20, 30]
print(lst[2])
```

```
30
```

[12]:
```
s = {10, 20, 30}
print(s[2])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [12], in <cell line: 2>()
      1 s = {10, 20, 30}
----> 2 print(s[2])

TypeError: 'set' object is not subscriptable
```

[13]:
```
lst = [10, 20, 30, 40, 10]
print(lst)
```

```
[10, 20, 30, 40, 10]
```

[14]:
```
lst = {10, 20, 30, 40, 10}
print(lst)
```

```
{40, 10, 20, 30}
```

[15]:
```
s = {10, 20, 140}
s.add(30)
print(s)
```

```
{10, 140, 20, 30}
```

[16]:
```
lst = [10, 20, 30, 40, 10, 20, 30, 40, 10]
s = set(lst)
print(s)
```

```
{40, 10, 20, 30}
```

[18]:
```
string = 'aaaabbbbccddddeeeff'
s = set(string)
print(s)
print(len(s))
```

```
{'a', 'd', 'b', 'e', 'f', 'c'}
6
```

[19]:
```
# set elements should be immutable
s = {[10, 20, 30], [10, 20, 30], [10, 20, 30]}
print(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [19], in <cell line: 2>()
      1 # set elements should be immutable
----> 2 s = {[10, 20, 30], [10, 20, 30], [10, 20, 30]}
      3 print(s)

TypeError: unhashable type: 'list'
```

[20]:
```
# set elements should be immutable
s = {(10, 20, 30), (10, 20, 30), (10, 20, 30)}
print(s)
```

```
{(10, 20, 30)}
```

2.1 Empty set creation

[21]:
```
l = []
print(type(l))
```

```
<class 'list'>
```

[22]:
```
s = ""
print(type(s))
```

```
<class 'str'>
```

[23]:
```
se = {}
print(type(se))
```

```
<class 'dict'>
```

[24]:
```
se = set()
print(type(se))
```

```
<class 'set'>
```

2.2 Comprehensions on sets

[26]:
```
lst = [11, 17, 45, 20, 10, 15, 10, 45, 48, 61, 20]
new_list = [i for i in lst if i % 5 == 0] # list comprehension
s = set(new_list)
print(s)
```

```
{10, 20, 45, 15}
```

[27]:
```
lst = [11, 17, 45, 20, 10, 15, 10, 45, 48, 61, 20]
new_set = {i for i in lst if i % 5 == 0} # set comprehension
print(new_set)
```

```
{10, 20, 45, 15}
```

2.3 set methods

2.3.1 set.add()

- To add a new element to an existing set

[30]:
```
s = {10, 20, 10}
s.add(30)
s.add(40)
print(s)
```

```
{40, 10, 20, 30}
```

2.3.2 set.union()

- Used to find the union of two sets

[31]:
```
s1 = {10, 20, 10} # {10, 20}
s2 = {20, 10, 20} # {20, 10} {10, 20}
s3 = s1.union(s2)
print(s3)
```

```
{10, 20}
```

[32]:
```
s1 = {'a', 'b', 'c', 'd'}
s2 = {'z', 'x', 'd', 'c'}
print(s2.union(s1)) # s1 union s2 <==> s2 union s1
```

```
{'a', 'z', 'd', 'x', 'b', 'c'}
```

2.4 set.difference()

- s1.difference(s2) gives the elements that are present in s1 but not in s2.
- s2.difference(s1) gives the elements that are present in s2 but not in s1.

[34]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
print(s1.difference(s2)) # present in s1 but not in s2
print(s2.difference(s1)) # present in s2 but not in s1
#s1.difference(s2) != s2.difference(s1)
```

```
{10, 20, 30}
{80, 60, 70}
```

2.4.1 set.intersection()

- Common elements in both sets
- s1.intersection(s2) is same as s2.intersection(s1)

[36]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
print(s1.intersection(s2))
print(s2.intersection(s1))
```

```
{40, 50}
{40, 50}
```

[38]:
```
lst = [10, 20, 30]
lst2 = [10, 40, 50]
print(set(lst).intersection(set(lst2)))
```

```
{10}
```

[42]:
```
# sorting a set
s1 = {'x', 'p', 'w', 'z', 'q'}
s2 = {'a', 'z', 'p', 'q', 't'}
s3 = s1.intersection(s2)
lst = list(s3)
lst.sort()
print(lst)
```

```
['p', 'q', 'z']
```

GCD of two given numbers using sets

[52]:
```
# gcd(12, 18)
# 12 --> 1 2 3 4 6 12
# 18 --> 1 2 3 6 9 18
# cd --> 1 2 3 6
# gcd --> 6
a = int(input())
factors_a = {i for i in range(1, a + 1) if a % i == 0}
b = int(input())
factors_b = {i for i in range(1, b + 1) if b % i == 0}
# print(factors_a)
# print(factors_b)
common_factors = factors_a.intersection(factors_b)
print(f'GCD of {a} and {b} is: {max(common_factors)}') # greatest of common␣
 ↪factors
```

```
12
18
GCD of 12 and 18 is: 6
```

2.4.2 set.symmetric_difference()

- s1.symmetric_difference(s2) is the elements present either in a or in b, but not in both a and b
- symmetric_difference of two sets can also be termed as the difference of thier union and intersection.

[53]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
# intersection --> common elements
s3 = s1.symmetric_difference(s2)
print(s3)
```

```
{80, 20, 70, 10, 60, 30}
```

[54]:
```
string1 = 'python'
string2 = 'jython'
print(set(string1).symmetric_difference(set(string2)))
```

```
{'j', 'p'}
```

2.4.3 set.update()

[58]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
s1.update(s2) # it is going to update s1 with s1.union(s2)
print(s1)
print(s2)
```

```
{70, 40, 10, 80, 50, 20, 60, 30}
{80, 50, 70, 40, 60}
```

2.4.4 set.difference_update()

[61]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
# s1.difference_update(s2) # updates s1 with s1.difference(s2)
s2.difference_update(s1) # updates s2 with s2.difference(s1)
print(s1)
print(s2)
```

```
{50, 20, 40, 10, 30}
{80, 70, 60}
```

2.4.5 set.intersection_update()

[62]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
s1.intersection_update(s2) # updates s1 with s1.intersection(s2)
print(s1)
print(s2)
```

```
{40, 50}
{80, 50, 70, 40, 60}
```

2.4.6 set.symmetric_difference_update()

[63]:
```
s1 = {10, 20, 30, 40, 50}
s2 = {50, 40, 70, 60, 80}
# updates s1 with s1.symmetric_differece(s2)
s1.symmetric_difference_update(s2)
print(s1)
print(s2)
```

```
{80, 20, 70, 10, 60, 30}
{80, 50, 70, 40, 60}
```

2.4.7 set.issubset()

- s1.issubset(s2) returns True if s1 is a subset of s2, else false
- A set s1 can be called as a subset of s2 if every element in s1 is is also present in s2.

[1]:
```
s1 = {10, 20, 30}
s2 = {10, 20, 30, 40, 50}
print(s1.issubset(s2))
```

```
True
```

[2]:
```
s1 = {10, 20, 30}
s2 = {10, 20, 30, 40, 50}
print(s2.issubset(s1))
```

```
False
```

2.4.8 set.issuperset()

- s1.issuperset(s2) returns True, if s1 is a superset of s2.
- A set s1 is called superset of another set s2, if every element in s2 is also present in s1.

[3]:
```
s1 = {10, 20, 30}
s2 = {10, 20, 30, 40, 50}
print(s2.issuperset(s1))
```

```
True
```

[4]:
```
s1 = {10, 20, 30}
s2 = {10, 20, 30, 40, 50}
print(s1.issuperset(s2))
```

```
False
```

2.4.9 set.pop()

- Used to pop an element from a set
- When used a random element will be popped from the set

[5]:
```
s1 = {10, 20, 30}
s1.add(40)
print(s1)
```

```
{40, 10, 20, 30}
```

[7]:
```
s1 = {10, 20, 30, 40, 50}
s1.pop()
print(s1)
s1.pop()
```

```
print(s1)
```

```
{20, 40, 10, 30}
{40, 10, 30}
```

2.4.10 set.remove()

- Removes a specifed element from the set, if it's a member of set
- Else raises a key error if element is not present in the set

[11]:
```
s1 = {10, 20, 30}
s1.remove(30)
print(s1)
```

```
{10, 20}
```

[12]:
```
s1 = {10, 20, 30, 40, 50}
s1.remove(100)
print(s1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [12], in <cell line: 2>()
      1 s1 = {10, 20, 30, 40, 50}
----> 2 s1.remove(100)
      3 print(s1)

KeyError: 100
```

2.4.11 set.discard()

- Removes an element from the set if it's a member
- Does nothing if the element is not a member

[13]:
```
s1 = {10, 20, 30, 40, 50}
s1.discard(40)
print(s1)
```

```
{50, 20, 10, 30}
```

[14]:
```
s1 = {10, 20, 30, 40, 50}
s1.discard(100)
print(s1)
```

```
{50, 20, 40, 10, 30}
```

[18]:
```
string = 'THIS IS PYTHON'
new_string = string.lower()
print(string)
```

```
print(new_string)
```

```
THIS IS PYTHON
this is python
```

[ ]:

[30]:
```
def pangrams(s: str) -> str:
    l = set(s.lower())
    l.discard(' ')
    if len(l) == 26:
        return "pangram"
    else:
        return "not pangram"
s = input()
print(pangrams(s))
```

```
A Quick BroWn fOX jumps ovEr the LAZY DOG
pangram
```

[29]:
```
s = 'aAbBCc'
print(set(s))
```

```
{'B', 'A', 'c', 'b', 'a', 'C'}
```