Tuple

In [1]:
```
#Creating a tuple
t=()
t1=(1,2,3,4)
t2=(4,"ram","shyam",2.5)
```

In [2]:
```
print(t)
```

```
()
```

In [3]:
```
type(t2)
```

Out[3]:
```
tuple
```

In [4]:
```
#Assessing tuple - Indexing and slicing
```

In [5]:
```
t1[0]
```

Out[5]:
```
1
```

In [6]:
```
t1[-1]
```

Out[6]:
```
4
```

In [7]:
```
#indexing and slicing is similar to lists
t2[0:3]
```

Out[7]:
```
(4, 'ram', 'shyam')
```

In [8]:
```
#Can we edit tuple like list?
t1(0)=100
```

```
  Cell In[8], line 2
    t1(0)=100
    ^
SyntaxError: cannot assign to function call here. Maybe you meant '==' instead of '='?
```

In [9]:
```
#no, because tuple is like strings, they are immutable
```

In [10]:
```
#operation in Tuple- add, membership, multiple (lets see few)
t1=(1,2,3,4)
t2=(5,6,7,8)
t3=("ram","shyam","hari","jim")
```

In [11]:
```
print(t1+t2)
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
```

In [12]:
```
print(t2+t3)
```

```
(5, 6, 7, 8, 'ram', 'shyam', 'hari', 'jim')
```

In [13]:
```
print(t1*t2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 print(t1*t2)

TypeError: can't multiply sequence by non-int of type 'tuple'
```

In [14]:
```
#but we can multiply in this way:
print(t1*3)
```

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

In [15]:
```
#membership operation
print(1 in t1)
```

```
True
```

In [17]:
```
#There are many methods in tuple but lets see few
max(t1)
```

Out[17]:
```
4
```

In [18]:
```
min(t2)
```

Out[18]:
```
5
```

In [21]:
```
tuple(zip(t1,t2))
```

Out[21]:
```
((1, 5), (2, 6), (3, 7), (4, 8))
```

Sets

In [4]:
```
#Creating sets
s={}
s1={1,2,3,4,5}
```

In [5]:
```
#but, if we create sets in this way, there will be error:
s2={1,2,"ram",{3,4}}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 2
      1 #but, if we create sets in this way, there will be error:
----> 2 s2={1,2,"ram",{3,4}}

TypeError: unhashable type: 'set'
```

In [6]:
```
#beacuse always remember set cannot contain sets inside because sets is muttable, it can only contain immputabl
#like:
s2={1,2,"ram",(3,4)}
```

In [7]:
```
type(s2)
```

Out[7]:
```
set
```

In [8]:
```
#Editing sets: lets try to change
s2[0]=100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 2
      1 #Editing sets: lets try to change
----> 2 s2[0]=100

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

In [9]:
```
#We can edit sets but we can add, and update and delete.
s2.add(888)
```

In [10]:
```
print(s2)
```

```
{1, 2, (3, 4), 888, 'ram'}
```

In [11]:
```
s2.update([3,4,5,7,8])  #lets see will it accept dupliacte values
```

In [12]:
```
print(s2)
```

```
{1, 2, 3, 4, (3, 4), 5, 7, 8, 888, 'ram'}
```

In [14]:
```
#yes for now because 3,4 is in bracket as tuple, if we add 888 again
s2.add(888)
```

In [15]:
```
print(s2)
```

```
{1, 2, 3, 4, (3, 4), 5, 7, 8, 888, 'ram'}
```

In [16]:
```
#it will onle give unique output.
```

In [17]:
```
#Now deleting sets elements
del s2[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 2
      1 #Now deleting sets elements
----> 2 del s2[0]

TypeError: 'set' object doesn't support item deletion
```

In [18]:
```
#will it support remove?
s.remove(888)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[18], line 2
      1 #will it support remove?
----> 2 s.remove(888)

AttributeError: 'dict' object has no attribute 'remove'
```

In [19]:
```
#it only support clear?
s.clear()
```

In [20]:
```
print(s)
```

```
{}
```

Set Operation

In [21]:
```
# Membership Test
1 not in s1
```

Out[21]:
```
False
```

In [22]:
```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
s1 | s2      #Union
```

Out[22]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [23]:
```
# Intersection(&)
s1 & s2
```

Out[23]:
```
{4, 5}
```

In [24]:
```
# Difference(-)
print(s1 - s2)
print(s2 - s1)
```

```
{1, 2, 3}
{8, 6, 7}
```

In [25]:
```
# Symmetric Difference(^)
s1 ^ s2
```

Out[25]:
```
{1, 2, 3, 6, 7, 8}
```

Python Cheat Sheet: Set Methods

"A puzzle a day to learn, code, and play" → Visit finxter.com

Method	Description	Example
`set.add(x)`	Add an element to this set	`>>> a = {1, 2, 3}` `>>> a.add(4)  # {1, 2, 3, 4}`
`set.clear()`	Remove all elements from this set	`>>> a = {1, 2, 3}` `>>> a.clear()  # set()`
`set.copy()`	Create and return a flat copy of this set	`>>> a = {1, 2, 'Alice'}` `>>> a.copy()  # Returns: {1, 2, 'Alice'}`
`set.difference(x)`	Return a new set with elements of this set except the ones in the given set arguments.	`>>> {1, 2, 3}.difference({1, 2})` `{3}`

`set.difference_update(iter)`	Remove all elements from this set that are members of any of the given set arguments.	`>>> a = {1, 2, 3}`
		`>>> a.difference_update({1, 2})`
		`# a == {3}`

`set.discard(x)`	Remove an element from this set if it is a member, otherwise do nothing.	`>>> a = {'Alice', 'Bob', 'Cloe'}`
		`>>> a.discard('Bob')`
		`# a == {'Alice', 'Cloe'}`

`set.intersection()`	Return a new set of elements that are members of this and the set argument(s).	`>>> {1, 2, 3, 4}.intersection({3, 4, 5})` `{3, 4}`

`set.intersection_update()`	Removes all elements from this set that are not members in all other specified sets.	`>>> a = {1, 2, 3, 4}`
		`>>> a.intersection_update({3, 4, 5})`
		`# a == {3, 4}`

`set.isdisjoint(x)`	Return True if their intersection is the empty set.	`>>> {1, 2, 3, 4}.isdisjoint({'Alice', 'Bob'})` `True`

`set.issubset()`	Return True if all elements of this set are members of the specified set argument.	`>>> t = {'Alice', 'Bob', 'Carl', 'Liz'}`
		`>>> {'Alice', 'Bob'}.issubset(t)`
		`True`

`set.issuperset()`	Return True if all elements of the specified set argument are members of this set.	`>>> {'Alice', 'Bob', 'Carl'}.issuperset({'Alice'})` `True`

`set.pop()`	Remove and return a random element from this set. KeyError if set is empty.	`>>> a = {'Alice', 'Bob', 'Carl'}`
		`>>> a.pop()`
		`'Alice'`

`set.remove()`	Remove and return a specific element from this set as defined in the argument. If the set doesn't contain element, raise KeyError.	`>>> a = {'Alice', 'Bob', 'Cloe'}`
		`>>> a.remove('Bob')`
		`# a == {'Alice', 'Cloe'}`

`set.symmetric_difference()`	Return new set with elements in either this or the specified set argument, but not both.	`>>> {1, 2, 3}.symmetric_difference({2, 3, 4})` `{1, 4}`

`set.symmetric_difference_update()`	Replace this set with the symmetric difference, i.e., elements in either this set or the specified set argument, but not both.	`>>> a = {1, 2, 3}`
		`>>> a.symmetric_difference_update({2, 3, 4})`
		`>>> a`
		`{1, 4}`

`set.union()`	Create and return new set with all	`>>> {1, 2, 3, 4}.union({3, 4, 5})` `{1, 2, 3, 4, 5}`

finxter

Sets methods:

Dictionary

In [26]:
```
d={}
```

In [27]:
```
type(d)
```

Out[27]:
```
dict
```

In [28]:
```
s={}
```

In [29]:
```
type(s)
```

Out[29]:
```
dict
```

In [30]:
```
d1={"city":"kathmandu", "person":"ram","salary":5000}
```

In [31]:
```
print(d1)
```

```
{'city': 'kathmandu', 'person': 'ram', 'salary': 5000}
```

In [35]:
```
print(d1.keys())
```

```
dict_keys(['city', 'person', 'salary'])
```

In [36]:
```
print(d1.values())
```

```
dict_values(['kathmandu', 'ram', 5000])
```

In [38]:
```
print(d1.items())
```

```
dict_items([('city', 'kathmandu'), ('person', 'ram'), ('salary', 5000)])
```

In [41]:
```
d1['city']
```

Out[41]:
```
'kathmandu'
```

In [46]:
```
#we can use get method too, is it same? yes output is same
d1.get("city")
```

Out[46]:
```
'kathmandu'
```

In [42]:
```
#Direct indexing and slicing with position like in List, tuples, strings; it is not possible in Sets and Dict.
```

In [43]:
```
#we can add/update in dictionary:
d1["age"]=35
```

In [44]:
```
print(d1)
```

```
{'city': 'kathmandu', 'person': 'ram', 'salary': 5000, 'age': 35}
```

In [45]:
```
#update way is also same
d1["age"]=55
print(d1)
```

```
{'city': 'kathmandu', 'person': 'ram', 'salary': 5000, 'age': 55}
```

In [50]:
```
#for loop in dictionary
for i in d1:
    print(i)
```

```
city
person
salary
age
```

In [52]:
```
for i,j in d1.items():
    print(i,j)
```

```
city kathmandu
person ram
salary 5000
age 55
```

In [53]:
```
for i,j in enumerate(d1.items()):
    print(i,j)
```

```
0 ('city', 'kathmandu')
1 ('person', 'ram')
2 ('salary', 5000)
3 ('age', 55)
```

Python dictionary methods: