PROGRAMMING WITH PYTHON: TUPLES & SETS

TUPLE

- A Tuple is a sequence of Python objects separated by commas.
- A tuple is similar to a list in terms of indexing, nested objects and repetition
- BUT a tuple is immutable unlike lists which are mutable.
  - TUPLE is immutable (There is a bit of ambiguity.)

Creating TUPLE

- The tuple is created by enclosing elements in parentheses (Optional)
  - **tup1 = (111, 12, 11)**
  - **tup2 = 111, 12, 11**

- A tuple with a single element must have a comma inside the parentheses:
  - **tup3 = (11,)**
    - **(11) without the comma is the integer 11**
    - **(11,) with the comma is a tuple containing the integer 11**

- An empty tuple can be created by empty parentheses:
  - **tup4 = ()**
  - **tup5 = tuple()**

tuple() constructor can be used to create tuple.
tup7 = tuple((1, 2, 3))

- A nested tuple can be created by having the tuple inside the tuple :
  - **tup6 = ((1,2), 3, 4, 'Keqin', 'Zahid', 'Raza', 'Haider')**

Creating TUPLE

tuple() constructor can be used to create tuple.
tuple(iterable/list/string/tuple/set/dict)

tup1 = tuple([1, 2, 3])
tup2 = tuple(‘Python’)
tup3 = tuple((1, 2, 3))
tup4 = tuple({1, 2, 3})
tup5 = tuple({1:2, 2:4, 3:8})

ADVANTAGES OF TUPLE OVER LIST

- **In general,** tuple is used for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, **iterating through tuple is faster than with list**. So there is a slight performance boost.
- **Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.**
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Tuple: Accessing Elements

- **To access values in tuples, use the square bracket with the index of the element.**

- Each item in a tuple has a unique index that specifies its position. The index of the first item is 0 and the index of the last item is the length of the list minus 1.

- The tuple index can be either positive, negative or an expression
  - Positive index starts from the leftmost element of tuple and it ranges from 0 to (length of tuple) minus 1.
  - Negative index starts from the rightmost element of tuple and it ranges from -1 to minus length of tuple (i.e. –length).

- The nested tuples are accessed using nested indexing.

Tuple: Accessing Elements

- tup1= ('Joseph', 'Glenn', 'Sally', ('Haider', 'Raza'), 1, 2, 3)
- **Positive Index**
  - tup1[0], tup1[1], tup1[2], tup1[3]
- **Negative Index**
  - tup1[-4], tup1[-3], tup1[-2], tup1[-1]
- **Arithmetic Expression as an Index**
  - tup1[20*6 - 19*3 - 60]
- **Nested tuple**
  - tup1[3][0], tup1[3][1]

Tuple Slices

- Tuple is sequential data type, slicing can be used.
- The slice of a tuple is the continuous piece of the tuple.
- To extract a contiguous piece of a tuple, use a subscript consisting of the starting position followed by a colon (:), finally followed by one more than the ending position of the slice you want to extract.
- **It is also possible to extract non-continuous elements of tuple.**
- If you omit the first index, the slice starts at the beginning.
- If you omit the second, the slice goes to the end.
- If you omit both, the slice is a copy of the whole tuple.
- If we omit first and second index and provide step value as -1, then the reverse of tuple will be created.

Tuple Slices

- tup1 = ('a', 'b', 'c', 'd', 'e', 'f‘)

  tup1[1:3] ---------- ('b', 'c‘)
  tup1[:4] ---------- ('a', 'b', 'c', 'd‘)
  tup1[3:] ---------- ('d', 'e', 'f‘)
  tup1[:] ---------- ('a', 'b', 'c', ‘d’, ‘e', 'f‘)
  tup1[0:6:2] ------- ('a‘, 'c‘, 'e’)
  tup1[::-1] --------- (‘f’, ‘e‘, ‘d‘, ‘c’ ,‘b‘, ‘a’)

Using the range function

- The range function returns an object of range class that can be converted to tuple using tuple() constructor.

```
print(tuple())
print(tuple(range(4)))  ------(0, 1, 2, 3)
print(tuple(range(1, 4)))  ------(1, 2, 3)
print(tuple(range(2, 10, 2))) ------(2, 4, 6, 8)
print(tuple(range(-1, -11, -2))) ------(-1, -3, -5, -7, -9)
```

Tuples are Immutable/Mutable

- **Strings are "immutable" - we *cannot* change the content of a string - we must create a new string for any change.**

- Lists are "mutable" - we *can* change its **state—its length and** content.

- Python tuples have a surprising trait:
  - They are immutable, but their values may be changed.
  - This may happen when a tuple holds a reference to any mutable object, such as a list.
    - The object at any index in tuple cannot be changed but if the referred object is mutable (Like a list), then, content of the mutable object can be changed within the tuple.

Tuples are Immutable/Mutable

Example:

```
A = ('Sajid', 1, [2, 3])
>>> id(A) ----  37076824
>>> A[0] ---- 'Sajid'
>>> A[0][0] ---  'S'
>>> A[0][0] = 'B‘ ----
    Traceback (most recent call last):
    File "<pyshell#69>", line 1, in <module>
    A[0][0] = 'B'
    TypeError: 'str' object does not support item assignment
>>> A[2] ---- [2, 3]
>>> A[2][0] ---- 2
>>> A[2][1] --- 3
>>> A[2].append(5)
>>> A ---- ('Sajid', 1, [2, 3, 5])
>>> id(A) ---  37076824
```

The length of tuple cannot be changed.

Adding and removing items are not possible

Tuples are Mutable/immutable

Updating/ Deletion single element

- If tuple element holds a reference to any mutable object, it can be update/deleted.
- If tuple element holds a reference to any immutable object, it cannot be update/deleted.

Basic Tuple Operations

Concatenation

- + : The + operator concatenates two tuple i.e. it appends the second tuple at the tail of first tuple.
- Example: (1, ‘a’, 2, 3) + (223, 23) will return (1, ‘a’, 2, 3, 223, 23)

Repetition

- * : The * operator is used to repeat the tuple multiple times.
- Example: (1, ‘a’, 2, 3)*3 will return (1, ‘a’, 2, 3, 1, ‘a’, 2, 3, 1, ‘a’, 2, 3)

Membership operators

- **in** /**not in**: The in operator is used to check the membership of an element in the tuple.
- Example: 2 in (1, ‘a’, 2, 3) will return TRUE

Deleting a Tuple

Delete Tuples

- To explicitly remove an entire tuple, just use the **del** statement.
- Example –

  tuple1 = (‘Maths', ‘America', 1407)

  del(tuple1)

Basic Tuple Methods

tup1 = (1,2,3,4)

Length : **len()**: This function takes a tuple as a parameter and returns the length of tuple (i.e. number of *elements* in the tuple). Example- **len(tup1), len((1,2,3,4))**

Maximum: **max()**: This function takes a tuple and returns the largest element of the tuple, **if possible**. Example: max(tup1), max((1,2,3,4, 5))

Minimum: min(): This function takes a tuple and returns the smallest element of the tuple, **if possible** . Example: min(tup1), min((34, 5, 7, 8))

Summation : **sum()**: This function takes a tuple and returns the sum of all elements of the tuple, **if possible**. Example: sum(tup1), sum((32, 45, 6))

Tuple Methods

```
>>> x = tuple()
>>> type(x)
 >>> dir(x)
>>>
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index']
```

Check: help(x)

Tuple Methods

A = (2, 1, 20, 20, 2, 45, 672, 4, 6, 8, 10, 12, 14, 16, 18)

A.count(obj): It returns count of how many times element obj occurs in tuple A.

A.index(obj): It returns the lowest index in tuple A that element obj appears

tuple() --- Convert an iterable (list, string, set, dictionary) to a tuple.

sorted(A) --- Take elements in the tuple and return a new sorted list (does not sort the tuple itself).

reversed(A) – returns a reversed object.

A TALE OF TWO LOOPS...
Accessing Elements of **Tuple** Through Loop

Tuple is iterable

```
friends = ('Joseph', 'Glenn', 'Sally‘)
for friend in friends :
   print('Happy New Year:', friend)

for i in range(len(friends)) :
   friend = friends[i]
   print('Happy New Year:', friend)
```

Packing and Unpacking of Tuple

When a tuple is created, it is as though the items in the tuple have been "packed" into the object:

tup1 = (1,2,3,4)

If this "packed" object is subsequently assigned to a new tuple, the individual items are "unpacked" into the objects in the tuple:

(a, b, c, d) = tup1

Note:

When unpacking, the number of elements in the tuple on the left of the assignment must equal the number of elements in tuple on the right side.

Python program to create the colon of a tuple.

```
import copy as c
#create a tuple
tup = ("Python", 50, 60, [], True, 97)
print('The tuple is', tup)
print('The identity of tuple is', id(tup))
#make a clone of a tuple using deepcopy() function
tup_clone = c.deepcopy(tup)
tup_clone[3].append(230)
print('The clone of tuple is', tup_clone)
print('The identity of clone of tuple is', id(tup_clone))
print('The tuple is', tup)
print('The identity of tuple is', id(tup))
```

Exercise

Write a Python program to unpack a tuple in several variables.

Write a Python program to find the repeated items of a tuple.
Write a Python program to find the index of an item of a tuple.
Write a Python program to remove an empty tuple(s) from a list of tuples.

SET

Sets

- A set is an unordered collection of data that is **iterable, mutable, and has no duplicate elements**.
- Python's set class represents the mathematical notion of a set.
- The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.
- The set data type is based on a data structure known as a hash table.

Sets

- **A set is an unordered collection of items.**
- **Every element must be unique and immutable.**
  - ***Duplicates are eliminated***
  - *Elements are* ***Immutable*** like tuples, integers, strings etc.
- However, the set itself is mutable.
  - It is possible to add or remove items to set.
  - We *can* change its **state—its length and content.**

- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

Set Creation

- A set is created by placing all items (elements) inside **curly braces {},** separated by comma or by using the **built-in function set().**
  - It can have any number of items and they may be of different types (integer, float, tuple, string etc.).
  - A set cannot have a **mutable element**, like list, set or dictionary, as its element.
  - **The empty set can also be created by built-in function set().**
  - The empty curly braces {} will be treated as empty dictionary in Python.

- **>>> set1 = {11, 11, 22}**
- **>>>set2---- {11, 22}**
- **>>>set3 = set() ---- { }**
- **>>> set4 = set({1, 2, 3})**

Set Creation
Using the range function

- The range function returns a set of numbers that consists of elements from zero to one less than the parameter.

print(set()) ------set(). **Why not {}**

print(set(range(4)) ------{0, 1, 2, 3}

print(set(range(1, 4)) ------{1, 2, 3}

print(set(range(2, 10, 2)) ------{2, 4, 6, 8}

print(set(range(-1, -11, -2)) ------{-1, -3, -5, -7, -9}

Sets have no order

```
>>>A = {1111111123, 222222222, 3333333333333, 4444444444444, 5555555555}
>>> A
{5555555555, 222222222, 1111111123, 3333333333333, 4444444444444}
```

SETS HAVE NO DUPLICATEs

```
>>>A = {1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5}
>>> A
{1, 2, 3, 4, 5}
```

SETS DON'T CONTAIN MUTABLEs

```
>>>A = {1, 2, 3, 4, 5, 'PYTHON', 2.3, [12, 34]}
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    A = {1, 2, 3, 4, 5, 'PYTHON', 2.3, [12, 34]}
TypeError: unhashable type: 'list'
```

SETS Don't SUPPORT INDEXING

- >>> myset = {'Apples', 'Bananas', 'Oranges'}
- >>> myset
  {'Bananas', 'Oranges', 'Apples'}
- >>> myset[0]
  Traceback (most recent call last):
  File "<pyshell#390>", line 1, in <module>
  myset[0]
  TypeError: 'set' object does not support indexing

So, How to access elements of set?

Sets are iterable
A tale of loop

```
friends = {'Joseph', 'Glenn', 'Sally‘}
for friend in friends :
  print('Happy New Year:', friend)
```

Sets are mutable

- Sets are mutable.
- add() : It takes one element and adds this element to the set.
- update(): It takes **string/set/tuple/list/dictionary** and adds all elements to the set.
- pop(): It removes one item from the set and returns it.
  - Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.
- remove() : It removes an element from a set. If the element is not a member, raise a KeyError
- clear(): This function remove all items from a set.

Example:

- >>> set1 = {11, 22, 33}
- >>> set1.add(2) ---- will add 2 to set1
- >>> set1.update([1, 5, 7]) will add 1, 5 and 7 to set1
- >>> set1.remove(5) --- It removes 5 from the set.
- >>> set1.pop() ---- It may remove any item.
- >>> set1.clear() --- It removes all item in a set

Basic Operations

Concatenation

- Not Supported

Repetition

- Not Supported

Membership operators

- **in /not in**:
- The in operator is used to check the membership of an element in the set.

2 in {1, 2, 3} will return **True**

4 in {1, 2, 3} will return **False**

SET Methods

```
>>>x = set()
>>> dir(x)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__',
'__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__',
'__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference',
'difference_update', 'discard', 'intersection', 'intersection_update',
'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference',
'symmetric_difference_update', 'union', 'update']
```

```
help(x)
```

Python Set Operations

- Sets can be used to carry out the set operations

A = {1, 2, 3, 4, 5, 6}

B = {5, 6, 7, 8, 9}

- Union
  - The union of A and B consists of all elements from both sets.
  - Union is performed using | operator. It can also be accomplished using the method union().
  - The | operator/union() method takes two set and returns a new set consisting of union of two sets.
  - Example
    - A | B
    - A.union(B)

Python Set Operations

- Sets can be used to carry out the set operations

A = {1, 2, 3, 4, 5, 6}

B = {5, 6, 7, 8, 9}

- Intersection
  - The intersection of A and B consists of all elements that are common in both sets.
  - It is performed using & operator or method intersection().
  - The & operator/intersection() method takes two set and returns a new set consisting of intersection of two sets.
  - Example
    - A & B
    - A.intersection(B)

Python Set Operations

- Sets can be used to carry out the set operations

A = {1, 2, 3, 4, 5, 6}

B = {5, 6, 7, 8, 9}

- Set Difference
  - The difference of A and B (i.e. A - B) consists of elements that are only in A but not in B.
  - Similarly, B - A is a set of element in B but not in A.
  - It is performed using minus(-) operator or difference() method.
  - The & operator/intersection() method takes two set and returns a new set consisting of difference of two sets.
  - Example
    - A - B
    - A.difference(B)

Python Set Operations

- Sets can be used to carry out the set operations

A = {1, 2, 3, 4, 5, 6}

B = {5, 6, 7, 8, 9}

- Symmetric Difference
  - The symmetric difference of sets A and B is a set of elements in both A and B except those that are common in both
  - A ^ B = (A | B) – (A&B)
  - The symmetric difference is performed using ^ operator or method symmetric_difference().
  - The ^ operator takes two set and returns a new set consisting of symmetric difference of two sets.
  - Example
    - A ^ B
    - A.symmetric_difference(B)

Set: Methods

- A.update(B) ---Update the set A with the union of A and B
- A.difference_update(B) ---Remove all elements of set B from set A
- A.symmetric_difference_update(B) ---Update a set with the symmetric difference of A and B
- A.intersection_update(B) --- Update the set with the intersection of A and B

- isdisjoint() ---- Return True if two sets have a null intersection
- issubset() ----Return True if another set contains this set
- issuperset() ----Return True if this set contains another set

- discard() ---Remove an element from set if it is a member. (Do nothing if the element is not in set)
- copy() ---- Return a copy of a set

Set Functions

- len(): Returns the length (the number of items) of the set.
- max(): Returns the largest item in the set, **if possible**.
- min(): Returns the smallest item in the set **if possible**.
- sorted(): Returns a new sorted list from elements in the set, **if possible**.
- sum(): Retruns the sum of all elements in the set, **if possible** .

FROZENET

- Frozenset is a new class that has the characteristics of a set, but its state cannot be changed once created.
- The frozensets can be created using the function frozenset().
- **The sets are mutable, frozensets are immutable sets.**
- **Advantage over sets**
  - **Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.**

FROZENET

Example:

- A = frozenset({1, 2, 3, 4})
- B = {1, 30, 25}
- C = A | B
- >>> type(C)
- <class 'frozenset'>

Check the followings

- frozenset('Zahid Raza')
- **frozenset([1,2])**
- **frozenset({1, 2, [12, 34, 56], 23})**
- frozenset((1, 2, 3))
- frozenset({1, 2, 3})
- frozenset({1:2, 3:4, 5:6})

Frozenset Methods

```
>>> x = frozenset ()
>>> dir(x)
>>> ['__and__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__',
'__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__xor__', 'copy', 'difference', 'intersection',
'isdisjoint', 'issubset', 'issuperset', 'symmetric_difference', 'union']
```

Check: help(x)

FROZENET

The frozenset supports

- union(), intersection(), difference(), symmetric_difference()
- isdisjoint(), issubset(), issuperset()
- copy()

- The frozenset are mutale. They don't support
  - add()
  - remove()
  - pop()
  - update()

Set Programs

Write a Python program to add and remove item(s) from set.

Write a Python program to remove an item from a set if it is present in the set.

Write a Python program to create an intersection, union difference and symmetric difference of two sets.

Write a Python program to create a shallow copy of sets.

Write a Python program to create an intersection, union difference and symmetric difference of two frozensets.

Thank You