Chp04B Tuples

References

1. https://www.w3schools.com/python/python_tuples.asp w3school Python Tutorial (tuples)

2. https://www.tutorialspoint.com/python3/python_tuples.htm Tutorialspoint (tuples)

3. *https://docs.python.org/3/tutorial/index.html Can also be obtained from Python manual*

4. *Learning Python, 5th (mark Lutz Oreilly 2013) – chapter 4 - Introducing Python Object Types*

5. *Python for Everybody (Free book) – chapter 10 Tuples*

Definition

- A tuple is a collection which is **unchangeable** (immutable)
- In Python tuples are written with round brackets () and separated by comma.

Example:

```
names=("Kimario","Asha","George", "Hamis")
print(names)
('Kimario', 'Asha', 'George', 'Hamis')
```

Note: Items in a tuple as it is in the list are not necessarily have the same type. **Example**:

```
tuple2=("Juma", 93.5, 34, True)
```

- To create a tuple with single element, add comma after that element: `tuple3=(23,) # add comma. If no comma, it will be treated as an integer and not a tuple!!`

tuple constructor

The tuple constructor can be used to create a tuple.

```
#Note the use of () enclosing the items
countries=tuple(("Tanzania","Kenya","Uganda","R
wanda","Burundi","Somalia"))
print(countries)
```

Results into:

```
('Tanzania', 'Kenya', 'Uganda', 'Rwanda',
'Burundi', 'Somalia')
print(tuple("abcd")) #('a', 'b', 'c', 'd')
print(tuple(range(4)))  # (0,1,2,3)
tuple3=tuple() #empty tuple
tuple4=()      #empty tuple
```

- Once a tuple is created, you cannot change its values (add new elements, delete elements, replace elements, or reorder the elements)
- i.e. tuples are immutable/**unchangeable**. If you try to change it, you will get: TypeError: 'tuple' object does not support item assignment

```
names=("Kimario","Asha", "George", "Hamis")
```

Following statement will cause a TypeError:

```
names[0]="Wema"   # TypeError
```

Note: if you don't enclose the sequence elements by (), a tuple will be created. Example:

```
seq=1,2,3,4,23,46 #same as seq=(1,2,3,4,23,46)
print(seq)        #(1,2,3,4,23,46)
```

- To access the tuple item, use index number. The first index is 0, the last index is **len(tuple) -1**.

```
names=("Kimario","Asha","George", "Hamis")
print(names[0])   # Kimario
print(names[3])    #Hamis
```

Slicing is also allowed:

```
print(names[0:3]) # first 3 names
```

- **Note**: Slicing in tuple works similar to lists and strings
- To determine the number of items in the tuple use **len** function: `len(names)`

- To check if item exist, use **in** keyword. It is case sensitive. Work similar to lists

Looping through a tuple

- To loop through a tuple, can use **for** loop.
  Example: Print all items in the tuple, one by one each on a separate line. Works similar to Lists.

chp04Bex01NewTuple

```
countries=tuple(("Tanzania","Kenya","Ugand
a","Rwanda","Burundi","Somalia"))
print(countries)
for country in countries: #(pythonic way)
    print(country)
print()
#alternatively
for index in range(len(countries)):
    print(countries[index])
```

tuple operations

tuples support all the same operations as lists, except those that change the contents of the list. Tuples support the following:

- Subscript indexing (for retrieving element values only)
- Methods such as **index** and **count**
- Built-in functions such as len, min, and max
- Slicing expressions
- The in and not in operators
- The + and * operators

Tuples do not support methods such as append, remove, insert, reverse, and sort.

index method

help("tuple") – display methods associated with tuple

help("tuple.method_name") – help for a method

- To get the index of the first occurrence of the element with the specified value, use **index** method: works similar to lists

```
names=("Kimario","Asha","George", "Hamis")
pos=names.index("Asha")
```

Note: ValueError: tuple.index(x): x not in tuple occurs if the item does not exist.

Solution: First check if the tuple contains the item you want to search as it was done with lists

count method

- The **count** method returns the number of occurrence of the specified value. If the item does not exist, it returns zero. Works similar to lists

```
tuple2=(1,3,2,5,6,7,2,3,6)
tuple2.count(1)   # 1 occurrence
tuple2.count(3)   # 2 occurrence
tuple2.count(7)   # 1 occurrence
tuple2.count(12)  # 0 occurrence
```

https://www.w3schools.com/python/python_ref_tuple.asp **Full list of tuple methods. Click on each link of the method to get corresponding examples**

del statement

You can delete the tuple completely using:

```
del tupleName
```

Example:

```
names=("Kimario","Asha","George",
"Hamis")
del names  # delete the whole tuple
  print(names) #this will cause an error
because the tuple no longer exists
```

Note: You can't delete an individual item in tuple (since tuple is immutable)

min and max methods

max and **min** built in functions returns item from the tuple with max and min value respectively. **sum** function can also be applied. Works similar to lists..

Example:

```
tuple1=(12, 34, 56, 12)
print(max(tuple1))      #56
print(min(tuple1))      #12
print(sum(tuple1))      #114
```

Concatenating tuples - I

- Can concatenate tuples using + to create a new tuple from other tuples

```
tuple1=(12,34,56)
tuple2=(78,89)
tuple3=tuple1+tuple2
print(tuple3)   #(12, 34, 56, 78, 89)
```

Note: concatenation works if both are tuples

Concatenating tuples - II

```
tuple1=(12,34,56)
print(f'id(tuple1): {id(tuple1)}')
#2293166791744
tuple2=(78,65,23)
tuple1=tuple1+tuple2
print(f'tuple1=tuple1+tuple2: {tuple1}')
#(12, 34, 56, 78, 65, 23)
print(f'id(tuple1) after concatenation):
{id(tuple1)}') #1738628681544
```

Note: tuple1 obtained by concatenating tuple1 and tuple2 has a different address (using **id()**)compared to the first tuple1 as shown above. **Ids** above might be different on different computers.

Tuple unpacking - I

Same technique of unpacking as it was used with lists also applies to tuples

Suppose: `tuple1=(23,45,67)`

We can unpack the tuple using:

```
x,y,z=tuple1
```

The values of x,y and z becomes: 23, 45 and 67 respectively.

Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence.

Tuple unpacking - II

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a **list**.

```
tuple1=(12,34,56)
x,*y=tuple1
print(x) #12
print(y) # [34, 56] - gives a list!!
```

Tuple mutability

chp04Bex03Mutability

```
a=(1,); b=a
print(f'a= {a}')           #a= (1,)
print(f'b= {b}')           #b= (1,)
print(f'id(a)= {id(a)}') #id(a)= 2499772379400
print(f'id(b)= {id(b)}') #id(b)= 2499772379400
a=a+(1,)
print(f'a= {a}')          # a= (1, 1)
print(f'a= {b}')          # b= (1,)
print(f'id(a)= {id(a)}')#id(a)=2499771618824 # new
print(f'id(a)= {id(b)}') #id(b)=2499772379400 #old
```

Converting between tuples and lists

- You can use the built-in `list()` function to convert a tuple to a list, and the built-in `tuple()` function to convert a list to a tuple. Example chp04Bex04Conversion

```
numberTuple = (1, 2, 3)
numberList = list(numberTuple)
print(numberList)
#Results: [1, 2, 3]
numberList = [1, 2, 3]
numberTuple = tuple(numberList)
print(numberTuple)
#Results: (1, 2, 3)
```

Why use tuples?

- difference between lists and tuples is immutability
- One reason that tuples exist is performance. Processing a tuple is faster than processing a list (due to Python's implementations) so tuples are good choices when you are processing lots of data, and that data will not be modified.
- Another reason is that tuples are safe. Because you are not allowed to change the contents of a tuple, you can store data in one and rest assured that it will not be modified (accidentally or otherwise) by any code in your program.
- Additionally, there are certain operations in Python that require the use of a tuple. As you learn more about Python, you will encounter tuples more frequently.

==End of chapter 4B ==

Next: Chp 4C Dictionary