Chapter-6 Python Tuples

Contents

Chapter-6 Python Tuples

Python တွင် tuple နှင့် list ဟူ၍ sequence structure နှစ်မျိုးရှိသည်။ Tuple နှင့် list တို့ထဲတွင် ရှိသည့် element များအားလုံးသည် Python object များ ဖြစ်ကြသည်။

Python တွင် tuple နှင့် list ကို object များ အစဉ်တကျ နေရာတကျ ထည့်ထားနိုင်သည့် ကွန်တိန်နာ (ordered container)များအဖြစ် မှတ်ယူနိုင်သည်။ Dictionary ထဲတွင် သိမ်းဆည်းသည့် ဒေတာများသည် အစီအစဉ်တကျ သိုလှောင်ထားခြင်းမရှိပါ။ (Dictionaries do not store information in any particular order)။ Tuple ၏ အဓိပ္ပာယ်(definition)မှာ ကော်မာဖြင့် ခွဲခြားထားသည့် တန်ဖိုး အစုအဝေး (collection of comma-separated values)တစ်ခု ဖြစ်သည်။

Tuple နှင့် list တို့တွင် တူညီသည့်အချက် များစွာ ရှိသည်။ သို့သော် မတူညီသည့် အဓိက အချက်မှာ tuple များသည် ပြင်၍ ပြောင်း၍မရသည့်(immutable) data structure ဖြစ်သည်။ Immutable ဆိုသည်မှာ အသစ် ထပ်ထည့်ခြင်း၊ ရှိပြီးသား item များကို ဖျက်ပစ်ခြင်း၊ ပြောင်းလဲခြင်း လုပ်ခွင့်မပြုပါ။ Tuple ကို ပြောင်းလဲ၍ မရသည့် အသေဖြစ်သည့်(constant) list တစ်မျိုးဟု သတ်မှတ်သည်။

ဥပမာ- နှစ် တစ်နှစ်အတွင်းရှိ လနာမည်များ၊ ရက်နာမည်များသည် မည်သည့်အခါမျှ ပြောင်းရန် လိုအပ် လိမ့်မည် မဟုတ်ပေ။ တစ်ဦးတစ်ယောက်ကြောင့် မှားယွင်းပြီး ပြင်ဆင်လိုက်ခြင်း မဖြစ်စေလိုသည့် အရာများ၊ ဒေတာများကို tuple အဖြစ် ပြုလုပ်ထားပြီး ကာကွယ်နိုင်သည်။

Tuple ထဲရှိ element များကို ထည့်(add လုပ်)၍ သော်လည်းကောင်း, ဖယ်ထုတ်(remove) လုပ်၍ သော် လည်းကောင်း, အစားထိုး(replace လုပ်)၍သော်လည်းကောင်း မရနိုင်ပါ။ သို့သော် tuple ထဲတွင်ရှိ နေသည့် element များ ကိုယ်တိုင်သည် mutable data အမျိုးအစားများ ဖြစ်နိုင်သည်။ ဥပမာ- tuple တစ်ခု ထဲတွင် list တစ်ခု ပါဝင် နေနိုင်သည်။ Tuple ထဲက list သည် mutable data အမျိုးအစား ဖြစ်သောကြောင့် ပြောင်းနိုင်ပြင်နိုင်သည်။ Tuple အသုံးပြုရသည် အဓိက အားသာချက်များမှာ

- မတူညီသည့် ဒေတာ အမျိုးအစားများ(different (heterogeneous) data types)အတွက် သင့်လျော်သည်။
- Tuple များသည် immutable ဖြစ်ခြင်းကြောင့် dictionary ၏ key (key for a dictionary)အဖြစ် အသုံးပြုနိုင်သည်။ Dictionary များအကြောင်းကို နောက်ပိုင်းအခန်းများတွင် ရှင်းပြထားသည်။
- Tuple များကို iterate လုပ်ခြင်းသည် list များကို iterate လုပ်ခြင်းထက် ပို၍ မြန်ဆန်သည်။
- ပြင်ရန် ပြောင်းရန် မလိုသည့် data များ ထည့်သည့်အခါ(passing လုပ်သည့်အခါ) tuple များကို အသုံးပြုခြင်း ပိုကောင်း၏။
- Tuple များသည် memory နေရာ အနည်းငယ်သာယူသည်။
- မှား၍ ဖျက်မိခြင်း ပြောင်းမိခြင်း မဖြစ်နိုင်ပါ။ (can’t clobber tuple items by mistake.)
- Tuple များကို dictionary keys အဖြစ် အသုံးပြုနိုင်သည်။
- Tuple များကို function argument အဖြစ်သုံး၍ function ထဲသို့ ထည့်ပေးနိုင်သည်။

- Tuple နှင့် list အမျိုးအစားတွင် ဘာမျှမရှိသည့် empty tuple နှင့် empty list တို့ ပြုလုပ်နိုင်သည်။

6.1 Tuple အသစ် ပြုလုပ်ခြင်း (Constructing Tuples)

Tuple အသစ်တစ်ခု ပြုလုပ် တည်ဆောက်နိုင်သည့် နည်းများ(creating new tuples)မှာ

(၁) လက်သည်းကွင်း() ထဲတွင် element များကို ကော်မာခံ၍ ထည့်ရေးပြီး tuple ပြုလုပ်နိုင်သည်။

(၂) tuple() function ကို အသုံးပြု၍ tuple ပြုလုပ်နိုင်သည်။

```
x = (1, 2, 3)
x = 1, 2, 3
x = 2,                          # the comma tells Python it's a tuple
print(x, type(x))
```

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

```
empty_tuple = ()                # empty tuple အသစ် တစ်ခု တည်ဆောက်သည်။
print(empty_tuple)
print(type(empty_tuple))        # type() ဖြင့် အမျိုးအစားကို စစ်ဆေးသည်။
```

```
()
<class 'tuple'>
```

```
x = (0,1,2,3,4,5,6,7,8,9)    # integer tuple အသစ်တစ်ခု တည်ဆောက်သည်။
x
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
pet1 =['dog', 'cat', 'parrot']
print(type(pet1))
pet2=tuple(pet1)    # ရှိပြီးသား list တစ်ခုကို tuple တစ်ခု အဖြစ်သို့ ပြောင်းသည်။
print(type(pet2))
```

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

Element တစ်ခုထက် ပိုများသည့် tuple ပြုလုပ်ရန်အတွက် လက်သည်းကွင်း ()ထဲတွင် element များကို ကော်မာခံ၍ ထည့်ရေးသည်။

```
pets=('dog', 'cat', 'parrot')
pets
```

```
('dog', 'cat', 'parrot')
```

လက်သည်းကွင်း()မပါဘဲ element ၏ နောက်တွင် ကော်မာ ထည့်ရေး၍လည်း tuple ပြုလုပ် နိုင်သည်။

```
pets = 'dog',
```

```
pets
```

```
('dog',)
```

```
x = 2,                  # the comma tells Python it's a tuple
print(x, type(x))
```

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

Element တစ်ခုထက်ပိုများပါက လက်သည်းကွင်း()မပါဘဲ ကော်မာ(comma) ခံရေး၍လည်း tuple ပြုလုပ်နိုင်သည်။

```
pets = 'dog', 'cat', 'parrot'
pets
```

```
('dog', 'cat', 'parrot')
```

လက်သည်းကွင်း(parentheses) မပါဘဲ နောက်ဆုံး element ၏ နောက်တွင် ကော်မာ(comma) ထည့်ရေး၍လည်း tuple ပြုလုပ်နိုင်သည်။ သို့သော် လက်သည်းကွင်း(parentheses) ထဲတွင် ထည့်ရေးခြင်းကြောင့် ဖတ်ရလွယ်သည်။ ပရိုရမ်က output ထုတ်ပေးသည့်အခါ နားလည်မှု မရှုတ်ထွေးစေရန် လက်သည်းကွင်း(parentheses) ထဲတွင် ထည့်၍ အမြဲဖော်ပြလေ့ရှိသည်။

```
a = tuple(m for m in range(8))
print(a)
b = tuple(i**2 for i in range(10) if i>4)
print(b)
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
[25, 36, 49, 64, 81]
```

6.2 Tuples as Records

Tuple များကို မှတ်တမ်း မှတ်ရာများကို သိမ်းဆည်းရန်အတွက် အသုံးပြုနိုင်သည်။ (Tuples as records)

Los Angeles လေဆိပ်၏ တည်နေရာ(Latitude and longitude)

```
lax_coordinates = (33.9425, -118.408056)
lax_coordinates
```

```
(33.9425, -118.408056)
```

တိုကျိုမြို့တော်၏ အချက်အလက်များ (Data about Tokyo)

```
city, year, pop, chg, area = ('Tokyo', 2003, 32450, 0.66, 8014)
```

6.3 Tuple Unpacking

Tuple အတွင်းရှိ item များကို တစ်ခုချင်းစီဖြစ်အောင် ပြုလုပ်ခြင်းသည် tuple unpacking ဖြစ်သည်။ Temporary variable မသုံးဘဲ tuple အတွင်းရှိ item များ၏ value များကို ဖလှယ်(exchange) ခြင်းကို tuple unpacking ဟုခေါ်သည်။

```
x = ['pig', 'cow', 'horse']
a, b, c = x
print(a, b, c)
```

```
pig cow horse
```

ပိုသည့်(excess) item များကို စုထည့်ခြင်း

`*args` ဖြင့်သည့် function parameter ကို အသုံးပြုသည်။ `range(5)` သည် [0, 1, 2, 3, 4] ဖြစ်သည်။ အောက်တွင် a ကို range မှ ပထမ item ၊ b ကို ဒုတိယ item အဖြစ် သတ်မှတ်ပြီး ကျန်သည့် ကိန်းများအားလုံး အတွက် * ကို သုံး၍ တစ်စုတည်ပေါင်းပြီး item တစ်ခု အဖြစ်သတ်မှတ်သည်။

```
a, b, *rest = range(5)
a, b, rest
```

```
(0, 1, [2, 3, 4])
```

```
a, b, *rest = range(3)
a, b, rest
```

```
(0, 1, [2])
```

```
a, b, *rest = range(2)
a, b, rest
```

```
(0, 1, [])
```

*** prefix**

Index နံပါတ် 0, 3 နှင့် 4 ကို သက်ဆိုင်ရာ a, c, d တို့ဖြင့် assign လုပ်ပြီး ကျန်ကိန်းများ အားလုံးကို *body ထဲသို့ ထည့်သည်။

```
a, *body, c, d = range(5)
a, body, c, d
```

```
(0, [1, 2], 3, 4)
```

```
*head, b, c, d = range(5)
head, b, c, d
```

```
([0, 1], 2, 3, 4)
```

Temporary variable မသုံးဘဲ tuple အတွင်းရှိ item များ၏ value များကို ဖလှယ်(exchange)ခြင်းကို tuple unpacking ဟုခေါ်သည်။ Password နှင့် icecream တို့၏ value များကို အပြန်အလှန်လဲသည်။

```
password = 'swordfish'
icecream = 'tuttifrutti'
password, icecream = icecream, password
password
```

```
'tuttifrutti'
```

```
icecream
```

```
'swordfish'
```

6.4 Nested Tuples

ပရိုဂရမ် output တွင် tuple များမှ element များကို လက်သည်းကွင်း(parentheses) ထဲတွင် ထည့်၍ အမြဲဖော်ပြရသည့် တခြားသောအကြောင်းမှာ nested tuple များကို ရှင်းလင်းထင်ရှားစွာ ဖတ်နိုင်ရန် ဖြစ်သည်။

```
pets=('dog', 'cat'), 'parrot'
pets
```

```
(('dog', 'cat'), 'parrot')
```

('dog', 'cat') သည် main tuple ထဲတွင် ပါဝင်နေသည့် subtuple တစ်ခုဖြစ်သည်။ Nested tuple ထဲတွင် tuple ပေါင်းများစွာ အကန့်အသတ်မရှိ ထည့်ထားနိုင်သည်။ သတိပြုရန်အချက်မှာ tuple အလွှာ ပေါင်းများစွာ ထည့်ထားခြင်းကြောင့် ဖတ်ရှုရန် နှင့် tuple တည်ဆောက်ထားပုံ(structure)ကို နားလည်ရန် ခက်ခဲလိမ့်မည်။

6.5 Integer Vs Tuple with one Element

ကိန်းတစ်လုံးတည်းရှိသည့်အခါ integer လား? tuple လား? ဟု အမှတ်မှားစရာ ဖြစ်နိုင်သည်။ x = 2 ဟုသိထားသည့်အခါ ထို x ၏ data type ကိုလည်း သတိပြုရန်လိုသည်။ အောက်က ဥပမာတွင် x သည် integer တစ်ခု ဖြစ်နိုင်သလို tuple တစ်ခုလည်းဖြစ်နိုင်သည်။ သို့သော် tuple ဖြစ်ကြောင်းထင်ရှားစေရန် လက်သည်းကွင်း() ထဲတွင် ဖော်ပြထားသည်။

```
x =(2)
print(x)
print(type(x))
```

```
2
<class 'int'>
```

```
x = (2,)
print(x)          # tuple ဖြစ်ကြောင်း လက်သည်းကွင်း() ထဲတွင်ထည့်သွင်း ဖော်ပြလိမ့်မည်။
print(type(x))
```

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

Tuple တွင် variable များစွာကို တပြိုက်နက် assign လုပ်နိုင်သည်။

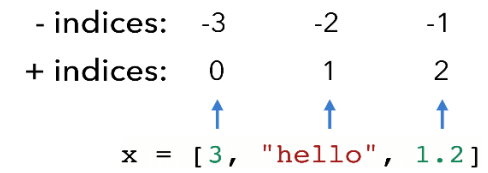
```
vehicle = ('Toyota', 'BMW', 'Benz')
a, b, c = vehicle
a, b, c
```

```
('Toyota', 'BMW', 'Benz')
```

6.6 Accessing Tuple Elements

Tuple အတွင်းရှိ element များကို နည်း နှစ်မျိုးဖြင့် access လုပ်နိုင်သည်။

- Indexing (element တစ်ခုကို access လုပ်လိုလျှင်)
- Slicing (တစ်ခုထက်ပိုများသည့် element များကို access လုပ်လိုလျှင်)



Indexing နည်းကို အသုံးပြု၍ tuple အတွင်းရှိ element များကို access လုပ်ခြင်း

```
pets = 'dog', 'cat', 'parrot'
pets[1]                                  # index နံပါတ် 2 ကို access လုပ်သည်
```

```
'cat'
```

မရှိသည့် element ကို access လုပ်လျှင် `IndexError: tuple index out of range` ပေးလိမ့်မည်။

```
pets[3]                          # IndexError: tuple index out of range
```

```
IndexError: tuple index out of range
```

အနောက်မှ စတင်၍ ပြန်လည် ညွှန်ပြလိုလျှင် negative index များကို အသုံးပြုသည်။ -1 သည် နောက်ဆုံးအလုံး(last element in the tuple) ဖြစ်သည်။ -2 သည် ဒုတိယ နောက်ဆုံးအလုံး(second from last element) ဖြစ်သည်။

```
pets = 'dog', 'cat', 'parrot', 'gerbil'
pets[-1]
```

```
'gerbil'
```

```
pets[-2]                                 # index နံပါတ် [-2]သည် ဒုတိယ နောက်ဆုံးအလုံး ဖြစ်သည်။
```

```
'parrot'
```

Tuple ၏ index နံပါတ်များ(list indices)သည် ကိန်းပြည့်များ(integers)သာ ဖြစ်ရမည်။ ကိန်းပြည့်များ(integers) မဟုတ်ခဲ့လျှင် TypeError ပေးလိမ့်မည်။

```
pets = 'dog', 'cat', 'parrot'
pets['1']                                # index နံပါတ်သည် string မဖြစ်ရပါ။
```

```
TypeError: tuple indices must be integers or slices, not str
```

Tuple သည် immutable ဖြစ်သော်လည်း tuple ထဲတွင် ရှိနေသည့် list သည် mutable ဖြစ်သည်။ (**tuples are immutable**, but member objects may be mutable.)

```
x = (1, 2, 3)
del(x[1])                    # Tuple ထဲက element ကို delete လုပ်လို့ မရပါ။
```

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

```
x[1] = 8                     # Tuple ထဲက element ကို  ပြောင်းလို့ မရပါ။
```

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

```
y = ([1, 2], 3)   # a tuple where the first item is a list
del(y[0][1])     # index နံပါတ်[0] သည် list ဖြစ်သည်။ ထိုထဲက index နံပါတ်[1]ကို delete လုပ်ရန်
print(y)        # the list within the tuple is mutable
```

```
([1], 3)
```

Tuple နှစ်ခုကို ဆက်ခြင်း(concatenating)

```
y += (4,)          # concatenating two tuples works
print(y)
```

```
([1], 3, 4)
```

6.7 Slicing

Python ရှိ list, tuple, str နှင့် တခြားသော sequence type များအားလုံးအတွက် slicing operationကို အသုံးပြုကြသည်။ **Slicing operation လုပ်သည့်အခါ နောက်ဆုံးအလုံး(last item)ကို ချန်ထားရသည့် အကြောင်းမှာ zero-based indexing ကြောင့် ဖြစ်သည်။**

[Start index: Stop index: Increment]

- Stop position ကို သုံးထားသောကြောင့် slice လုပ်မည့် အရေအတွက်(length of a slice)ကို အလွယ်တကူ သိနိုင်သည်။ (easy to see the length of a slice or range when only the stop position is given)။ ဥပမာ range(3) နှင့် my_list[:3] တို့တွင် item (၃)ခုပါရမည့်ဟု မြင်ရုံနှင့် တန်းသိနိုင်သည်။
- Slice တစ်ခု သို့မဟုတ် range တစ်ခုတွင် ပါဝင်နေမည့် အရေအတွက်ကို start နေရာနှင့် stop နေရာကို ကြည့်၍ အလွယ်တကူတွက်နိုင်သည်။ Item အရေအတွက် ကိုသိလိုလျှင် အဆုံး နေရာ မှ အစနေရာ ကို နုတ်ပါ။ **(Stop index - Start index)**
- Sequence တစ်ခုကို အပိုင်း (၂)ပိုင်း အဖြစ် အလွယ်တကူ ခွဲထုတ်ပစ်နိုင်သည်။

```
l = [10, 20, 30, 40, 50, 60]
l[:2]                              # split at 2
```

```
[10, 20]
```

```
l[2:]
```

```
[30, 40, 50, 60]
```

```
l[:3]                                  # split at 3
```

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

```
l[3:]
```

```
[40, 50, 60]
```

6.7.1 Slice Objects

Start index ၊ Stop index ၊ Increment တို့တွင် တန်ဖိုးများကို ဘာမှ မထည့်ပေးခြင်းသည် item အားလုံးကို ရည်ညွှန်းသည်။ ဘာမှ မပြောလျှင် increment သည် 1 ဖြစ်သည်။ (increment= 1)

```
s = tuple(range (0,10))
s[ : : ]                               # element များ အားလုံး( အစ မှ အဆုံးအထိ)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
s =  tuple(range (0,10))     # Item (၃)ခု မြောက် item တိုင်း
s[::3]
```

```
(0, 3, 6, 9)
```

```
s[::-1]                              # နောက်မှ စ, သည်။ ပြောင်းပြန် ပြန်စီသည်။
```

```
(9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
```

```
# နောက်မှ စ, ၍ ပြောင်းပြန် ပြန်စီသည်။ (၂)ခု မြောက် item တိုင်း ကိုယူရန် ။
s[::-2]
```

```
(9, 7, 5, 3, 1)
```

6.8 Accessing Tuple Elements Using Slicing

Indexing သည် element တစ်ခုချင်းစီ(individual element) ကိုသာ access လုပ်နိုင်သည်။ တစ်လုံးထက် ပိုများသည့် element များ(subset များ)ကို access လုပ်လိုသည်အခါ slicing ကို အသုံးပြု ရသည်။ Syntax မှာ

Tuple To Slice **[Start index (included) : Stop index (excluded) : Increment]**

: သည် slicing operator ဖြစ်သည်။

- **Start index:**
  The index at which to start the slicing. The element at this index is included in the

slice. If this parameter is absent, it is assumed to be zero, and thus, the slicing starts at the beginning of the tuple.

- **Stop index**:
  The index at which to stop slicing. The element at this index is not included in the slice. This means that the last item in this slice will be the one just before the stop index. If this parameter is absent, the slice ends at the very end of the tuple.
- **Increment:**
  This determines how many steps to take in the tuple when creating the slice. If this parameter is absent, it is assumed to be one.

Fruits အမည် ဖြင့် tuple တစ်ခုတည်ဆောက်ပြီး လုပ်ပုံကို ဖော်ပြထားသည်။

```
fruits = ("apple", "banana", "cherry", "orange", "kiwi", "melon",
          "mango")
print(fruits)
```

```
('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')
```

```
# return last element in the list
print(fruits[-1])            # [-1]သည် list ထဲမှ နောက်ဆုံးအလုံး
```

```
mango
```

```
# return elements from index 2 to 4 (index 5 is not included )
print(fruits[2:5])           # [2:5] သည် index နံပါတ် 2 မှ 4 အထိ (5 မပါ)
```

```
('cherry', 'orange', 'kiwi')
```

```
# returns the elements from 0 to 3 (prints first four elements )
print(fruits[:4])            # [ :4] သည် အစမှ index နံပါတ် 3 အထိ (4 မပါ)
```

```
('apple', 'banana', 'cherry', 'orange')
```

```
# returns the elements from index 2 to end
print(fruits[2:])            # [2: ] သည် index နံပါတ် 2 မှ အဆုံး အထိ
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

```
# return elements index -4 to -2(1- is not included)
print(fruits[-4:-1])   # [-4:-1]သည် index -4 မှ -2 အထိ (-1 မပါ)
a = fruits[2:5]
print(type(a))
```

```
('orange', 'kiwi', 'melon')
<class 'tuple'>
```

for loop နှင့် range() ကို သုံး၍ tuple တစ်ခုတည်ဆောက်ခြင်း

```
numbers = tuple([i for i in range(1,11)])
numbers
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

စုံကိန်း(even numbers)များသာ ပါသည့် tuple တစ်ခု တည်ဆောက်ရန်

```
even = numbers[1::2]
even
```

```
(2, 4, 6, 8, 10)
```

မကိန်း(odd number)များသာ ပါသည့် tuple တစ်ခု တည်ဆောက်ရန်

```
odd = numbers[::2]
odd
```

```
(1, 3, 5, 7, 9)
```

List ကို tuple ကို အဖြစ်သို့ ပြောင်းနိုင်သည်။ List နှင့် tuple တို့သည့် data structure များ ဖြစ်ကြသည်။

```
x = (0, 1, 2, 3, 4)
print(type(x))                    # Tuple  ဖြစ်ကြောင်း စစ်ရန်
```

```
<class 'tuple'>
```

```
x = list(x)                       # 'tuple' မှ 'list' အဖြစ်သို့ ပြောင်းသည်။
print(type(x))                    # List  ဖြစ်ကြောင်း စစ်ရန်
```

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

```
list1 = [2, 4, 6]
y = tuple(list1)
print(y, type(y))                  # 'tuple' အဖြစ်သို့ ပြောင်းသည်။
```

```
(2, 4, 6) <class 'tuple'>
```

```
x = (1, 2, 3)
print(x)
```

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

6.9 TypeError

x သည် tuple ဖြစ်သောကြောင့် delete လုပ်၍ မရပါ။ Delete လုပ်လျှင် TypeError ပေးလိမ့်မည်။

```
del(x[1])                         # TypeError
```

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

x သည် tuple ဖြစ်သောကြောင့် item assignment လုပ်၍ မရပါ။ လုပ်လျှင် TypeError ပေးလိမ့်မည်။

```
x[1] = 8                        # TypeError
```

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

6.10 Tuple Functions

Tuple သည် immutable ဖြစ်သောကြောင့် index method နှင့် count method သာ ရှိသည်။ ထို Tuple method နှစ်ခုသည် list တွင်ရှိသည့် index method နှင့် count method အလုပ်လုပ်ပုံနှင့် လုံးဝ တူညီသည်။

6.11 Tuple Methods

Python တွင် tuple နှင့် သက်ဆိုင်သည့် method အနည်းငယ်သာရှိသည်။ **any()** method သည် tuple အတွင်းရှိ element များသည် iterable element များ ဟုတ် မဟုတ် စစ်ပေးသည်။

```
pets = ('cat', 'dog', 'horse', 'dog', 'horse', 'horse')
any(pets)
```

```
True
```

count() method သည် tuple အတွင်း၌ သိလိုသည့် item များ အကြိမ်အရေအတွက်ကို ရေတွက် ပေးသည်။ တစ်နည်းအားဖြင့် သိလိုသည့် item တစ်ခုသည် tuple အတွင်း၌ အကြိမ် မည်မျှ ပါဝင်နေသည်ကို သိလိုသည့် အခါ အသုံးပြုသည်။ `tuple.count(element)`

```
pets = ('cat', 'dog', 'horse', 'dog', 'horse', 'horse')
pets.count("cat")
```

```
1
```

min() method သည် tuple အတွင်း၌ အကြိမ်အရေအတွက် အနည်းဆုံးပါဝင်နေသည့် element ကို ဖော်ပြပေးသည်။

```
pets = ('cat', 'dog', 'horse', 'dog', 'horse', 'horse')
min(pets)
```

```
'cat'
```

max() method သည် tuple အတွင်း၌ အကြိမ်အရေအတွက် အများဆုံးပါဝင်နေသည့် element ကို ဖော်ပြပေးသည်။

```
pets = ('cat', 'dog', 'horse', 'dog', 'horse', 'horse')
max(pets)
```

```
'horse'
```

len() method သည် tuple အတွင်း၌ ပါဝင်နေသည့် element များ၏ အရေအတွက်(total number of elements) ကို ဖော်ပြပေးသည်။

```
pets =('cat', 'dog', 'horse', 'dog', 'horse', 'horse')
```

```
len(pets)
```

```
6
```

Tuple များသည် string ကဲ့သို့ ပေါင်းစပ်(concatenate)ခြင်း ပြုလုပ်နိုင်သည်။

```
pets = ('cat', 'dog', 'horse')
wild = ('lion', 'zebra', 'antelope')
animals = pets + wild
print(animals)
```

```
('cat', 'dog', 'horse', 'lion', 'zebra', 'antelope')
```

if statement နှင့် **in** keyword ကို သုံးထားသည့် tuple ဥပမာ-

```
zoo = ("monkey", "liger", "quokka", "elephant")
```

zoo ဆိုသည့် list ထဲတွင် "elephant" ဆိုသည့် element ရှိမရှိ if ဖြင့် စစ်သည်။

```
if "elephant" in zoo:
    print("Yes, there is an elephant in your zoo.")
else:
    print("No, but there should be")
```

```
Yes, there is an elephant in your zoo.
```

6.12 Multidimensional Tuples

Tuple တစ်ခုထဲတွင် တခြားသော tuple များ အထပ်လိုက် ထားနိုင်သည်။ Multidimensional tuple ဟုခေါ်သည်။ Multidimensional tuple သည် multidimensional list နှင့် လုံးဝသဘောတရားနှင့် အလုပ်လုပ်ပုံ တူညီသည်။ Multidimensional list ကို list အခန်းတွင် ရှင်းပြခဲ့ပြီး ဖြစ်သည်။

Python list နှင့် tuple တို့၏ method အမျိုးမျိုးကို လက်တွေ့ အသုံးချနိုင်သည် အဆင့်ထိ ကြိုးစားလေ့ကျင့်သင့်သည်။ list နှင့် tuple တို့ကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်လျှင် ပရိုဂရမ်ရေးသည့်အခါ ပိုမြန်ဆန် အဆင်ပြေပြီး အမှားနည်းလာလိမ့်မည်။

- End –