Programming Fundamentals Lab

Lab # 11

Tuples and Dictionaries

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL1002

Semester Fall 2021

Department of Computer Science,

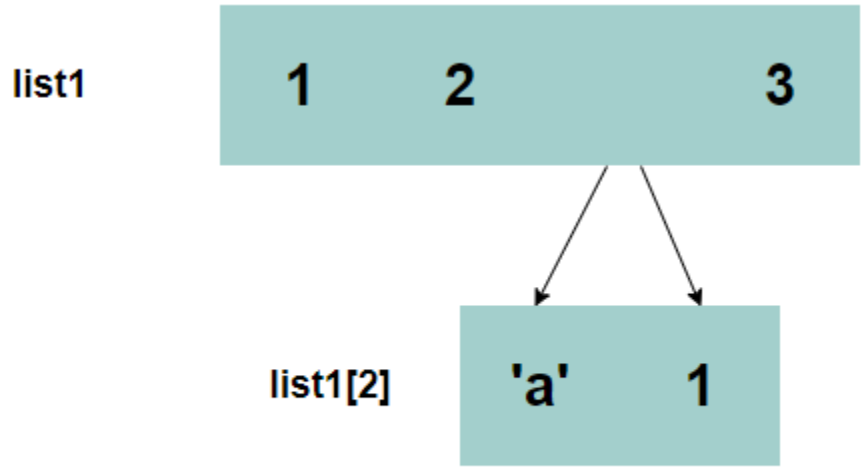
National University of Computer and Emerging Sciences FAST
Peshawar Campus

Contents

Nested List / List of List

A **nested list** is a *list of lists*, or any list that has another list as an element (a sublist). They can be helpful if you want to create a matrix or need to store a sublist along with other data types.

Lists can contain elements of different types, including other lists, as illustrated here:



An example of a nested list

Here is how you would create the example nested list above:

In [1]:

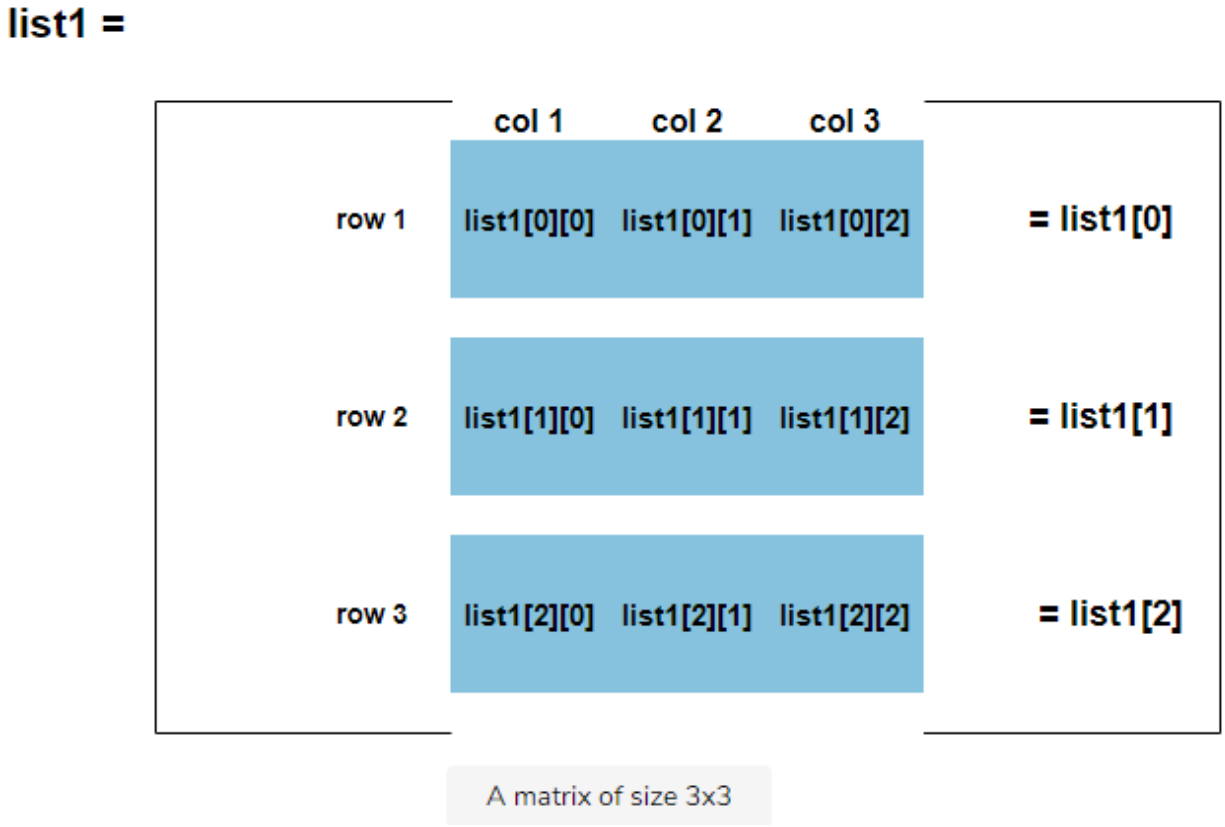
```
# creating list
nestedList = [1, 2, ['a', 1], 3]

# indexing list: the sublist has now been accessed
subList = nestedList[2]

# access the first element inside the inner list:
element = nestedList[2][0]

print("List inside the nested list: ", subList)
print("First element of the sublist: ", element)
```

```
List inside the nested list:  ['a', 1]
First element of the sublist:  a
```

Creating a matrix

Creating a matrix is one of the most useful applications of nested lists. This is done by creating a nested list that only has other lists of equal length as elements.

Each element of the nested list (matrix) will have two indices: the row and the column.



A matrix of size 3x3

In [11]:

```python
# create matrix of size 2 x 3
matrix = [
          [0, 1, 2],
          [3, 4, 5]
         ]

rows = len(matrix)  # no of rows is no of sublists i.e. len of list
cols = len(matrix[0])  # no of cols is len of sublist

# printing matrix
print("matrix: ")

for i in range(0, rows):
    print(matrix[i])

print("\nNumber of rows ",rows)
print("Number of columns ",cols)

# accessing the element on row 2 and column 1 i.e. 3
print("\nelement on row 1 and column 0: ", matrix[1][0])

# accessing the element on row 3 and column 2 i.e. 2
print("element on row 0 and column 2: ", matrix[0][2])
```

```
matrix:
[0, 1, 2]
[3, 4, 5]

Number of rows  2
Number of columns  3

element on row 1 and column 0:  3
element on row 0 and column 2:  2
```

Nested for Loop

In [13]:
```
#nested for loop
matrix = [
          [0, 1, 2],
          [3, 4, 5]
         ]

for i in range(2):
    for j in range(3):
        print(matrix[i][j])
```

```
0
1
2
3
4
5
```

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list

Syntax:

```
[expression for item in list]
```

In [28]:
```
#Using List comprehension
digits= [d for d in range(0,10)]
print(digits)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [33]:
```
#Using For Loop
digits=[]
for i in range(0,10):
    digits.append(i)
print(digits)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

If-else Statement:

In [45]:
```
#USING FOR LOOP
list1=[0,1,2,3,4,5,6,7,8,9]
evens = []
for i in list1:
   if i % 2 == 0:
      evens.append(i)
print(evens)
```

```
[0, 2, 4, 6, 8]
```

In [44]:
```
#USING List Comprehension
list1=[0,1,2,3,4,5,6,7,8,9]
evens = [i for i in list1 if i % 2 ==0]
print(evens)
```

```
[0, 2, 4, 6, 8]
```

In [48]:
```
#USING FOR LOOP
list1=[0,1,2,3,4,5,6,7,8,9]
eve_odd = []
for i in list1:
   if i % 2 == 0:
      eve_odd.append("Even")
   else:
    eve_odd.append("Odd")
print(eve_odd)
```

```
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

A better and faster way to write the above code is through list comprehension.

In [50]:
```
#USING List Comprehension
list1=[0,1,2,3,4,5,6,7,8,9]
obj = ["Even" if i%2==0 else "Odd"  for i in list1 ]
print(obj)
```

```
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

Tuple

A tuple is a collection which is **unchangeable**. In Python tuples are written with round brackets. It is also known as constant list.

Create Tuple

In [51]:
```
a=()
print(a)
```

```
()
```

In [52]:
```
a=("a","b")
print(a)
```

```
('a', 'b')
```

Accessing Tuple Item

You can access tuple items by referring to the index number:

In [55]:
```
a=("a","b")
print(a[1])
```

```
b
```

Multiple assignment through tuple

You can do multiple assignment using tuple

In [54]:
```
t=(1,2,3)
a,b,c=t
print("Value of a:",a)
print("Value of b:",b)
print("Value of c:",c)
```

```
Value of a: 1
Value of b: 2
Value of c: 3
```

Loop through Tuple

In [57]:
```
#loop through tuple
thistuple = ("apple","cherry","mango")
for i in thistuple:
    print(i)
```

```
apple
cherry
mango
```

Change, Add and Remove Tuple Item

Once a tuple is created, you cannot change it's values. Tuples are **unchangeable**.

```
:  1 #loop through tuple
   2 thistuple = ("apple","cherry","mango")
   3 thistuple[1]="orange"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-eca81333b9ce> in <module>
      1 #loop through tuple
      2 thistuple = ("apple","cherry","mango")
----> 3 thistuple[1]="orange"

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

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

```
:  1 #loop through tuple
   2 thistuple = ("apple","cherry","mango")
   3 del thistuple
   4 print(thistuple)  # this will raise an error because the tuple no longer exist
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-767141890dac> in <module>
      2 thistuple = ("apple","cherry","mango")
      3 del thistuple
----> 4 print(thistuple)  # this will raise an error because the tuple no longer exist

NameError: name 'thistuple' is not defined
```

Basic Function use on Tuple

1. **Whether item is in tuple or not:**

```
:  1 #loop through tuple
   2 thistuple = ("apple","cherry","mango")
   3 if "apple" in thistuple:
   4     print("Yes,'apple' is in the fruits tuple")
   5
```

```
Yes,'apple' is in the fruits tuple
```

2. Count

```
num_tuple=(1,3,7,8,7,5,4,6,8,5)
x=num_tuple.count(5)
print(x)
```

```
2
```

3. Length

```
num_tuple=(1,3,7,8,7,5,4,6,8,5)
x=len(num_tuple)
print(x)
```

```
10
```

4. Index

```
num_tuple=(1,3,7,8,7,5,4,6,8,5)
x=num_tuple.index(8)
print(x)
```

```
3
```

Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Dictionaries are used to store data values in key: value pairs.

Create

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique

```
In [65]:
1 thisdict = {}
2 print(thisdict)
```

```
{}
```

```
In [66]:
1 thisdict = {
2   "brand": "Ford",
3   "model": "Mustang",
4   "year": 1964
5 }
6 print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Accessing Item

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
In [67]:
1 x=thisdict["model"]
2 print(x)
```

```
Mustang
```

You can also access the items of dictionary using get()

```
x=thisdict.get("model")
print(x)
```

```
Mustang
```

Change or Add values

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

Change:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict['year'] =2018
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Add:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict['color'] = 'black'
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'black'}
```

Remove values

Here are several methods to remove items from a dictionary:

1. Pop()

```
In [12]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         thisdict.pop("model")
         print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

2. Popitem()

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
1 thisdict = {
2   "brand": "Ford",
3   "model": "Mustang",
4   "year": 1964
5 }
6 thisdict.popitem()
7 print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

3. Del

The del keyword removes the item with the specified key name:

```
In [14]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         del thisdict["model"]
         print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

The del keyword can also delete the dictionary completely:

```
In [15]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         del thisdict
         print(thisdict)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-45a6f6e38541> in <module>()
      5 }
      6 del thisdict
----> 7 print(thisdict)

NameError: name 'thisdict' is not defined
```

4. Clear()

The clear() keyword empties the dictionary:

```
In [16]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         thisdict.clear()
         print(thisdict)
```

```
{}
```

Loop through Dictionary

You can loop through a dictionary by using a `for` loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

1. Print all key names in the dictionary, one by one:

```
In [17]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         for x in thisdict:
             print(x)
```

```
brand
model
year
```

2. Print all *values* in the dictionary, one by one:

```
In [18]: for x in thisdict:
             print(thisdict[x])
```

```
Ford
Mustang
1964
```

3. You can also use the values() function to return values of a dictionary:

```
In [19]: for x in thisdict.values():
             print(x)
```

```
Ford
Mustang
1964
```

4. Loop through both *keys* and *values*, by using the items() function:

```
In [20]: for x, y in thisdict.items():
             print(x, y)
```

```
('brand', 'Ford')
('model', 'Mustang')
('year', 1964)
```

Basic Function use on Dictionaries

1. Whether item is in Dictionary or not:

To determine if a specified key is present in a dictionary use the `in` keyword:

```
In [23]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         if "model" in thisdict:
             print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

2. Len()

```
In [25]: thisdict = {
           "brand": "Ford",
           "model": "Mustang",
           "year": 1964
         }
         len(thisdict)
```

```
Out[25]: 3
```