Programming Fundamentals

Lecture 9 – Python Collections

Iresh Bandara

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

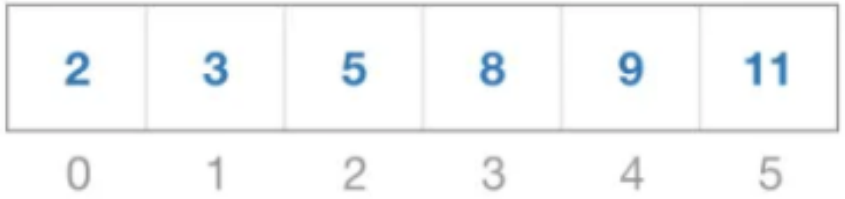
- This lecture addresses LO1,LO2 and LO4 for the module
- On completion of this lecture, students are expected to explain and apply
  - Python arrays, lists, tuples, sets and dictionaries
  - Relevant inbuilt functions needs to use for each situation
- Apply correct structure for a given task
- Compare the similarities and differences of each.

Agenda

- Python Arrays
- Python List – Introduction
- List – Accessing elements/ looping
- List Vs String
- List, List content and Aliases
- List – predefined methods
- Tuples
- Sets
- Dictionaries

Python Arrays

- data structure that contains a group of element
- can hold only homogeneous data (same type)
- Need to specify the type when creating the array.



Python Arrays

Type Code	C Type	Python Type	Minimum Size In Bytes
'b'	signed cahar	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

Python Arrays

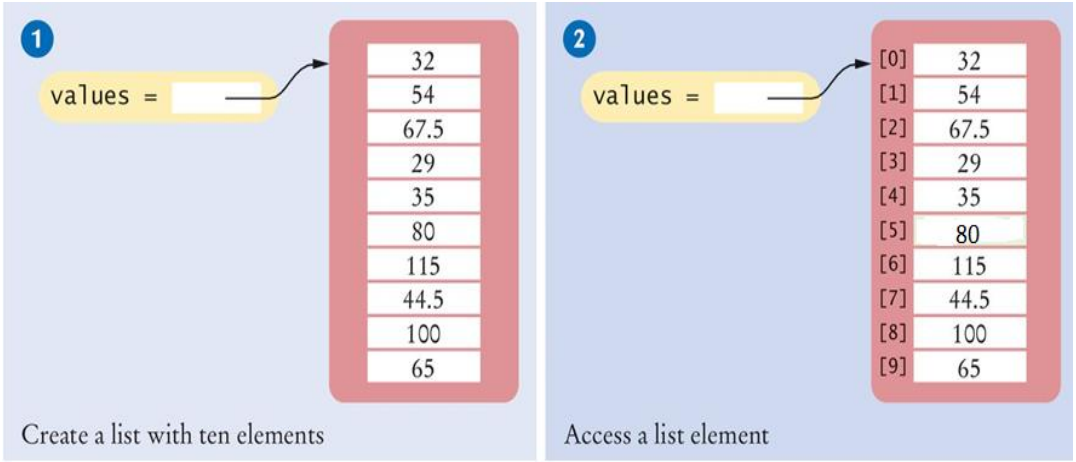
```
import array as arr

a = arr.array('d', [1.1, 3.5, 4.5,6.7,78,4.56,7.2])

print(a)

#print elements using index

print("First element:", a[0])

print("Second element:", a[1])

print("Last element:", a[-1])


#slicing arrays
print(a[2:5]) # 3rd to 5th
print(a[:-5]) # beginning to 4th
print(a[5:])  # 6th to end
print(a[:])   # beginning to end
```

```
# changing first element
a[0] = 0
print(a)

# changing 3rd to 5th element
a[2:5] = arr.array('d', [4, 6, 8])
print(a)
#append
a.append(100.67)
print(a)

#delete an index
del a[0]
print(a)

a.remove(100.67)
a.pop(3) #delete a given inedex
print(a)
```

Python lists - Introduction

- List is a collection of ordered, changeable items.
- Allow duplicate items and various types (int,float,String in same list
- Creating and accessing a list

```
values=[32,54,67.5,29…..]
print(values[1])
# print 54
print(values)
# print [32,54,67.5….]
```



Accessing range of indexes

- Accessing a range of items is similar to String slicing. You need to provide starting and end point indexes of the list.
- But starting point and end point is optional

```
items=[1,2,3,40,50]
print(items[1:4])
#print 2,3,40
```

```
items=[1,2,3,40,50]
print(items[:4])
#print 1,2,3,40
```

```
items=[1,2,3,40,50]
print(items[1:])
#print 2,3,40,50
```

Loop throughout the list

- WHILE or FOR can be used to iterate throughout the list.
- But FOR is the best choice as it does the auto increment.

```
items=[1,2,3,40]
for x in items:
    print(x)
```

```
items=[1,2,3,40]
counter=0
while counter<len(items):
  print(items[counter])
  counter += 1
```

List Vs Strings

- Both lists and strings are **sequences**, and the [] operator is used to access an element in any sequence
- There are two differences between lists and strings:
  - Lists can hold values of any type, whereas strings are sequences of characters
  - Strings are *immutable* - you cannot change the characters in the sequence:

```
greeting = "Hello, world!"
greeting[0] = 'J'              # ERROR! Check the Strings lecture
```

  - Lists are *mutable:*

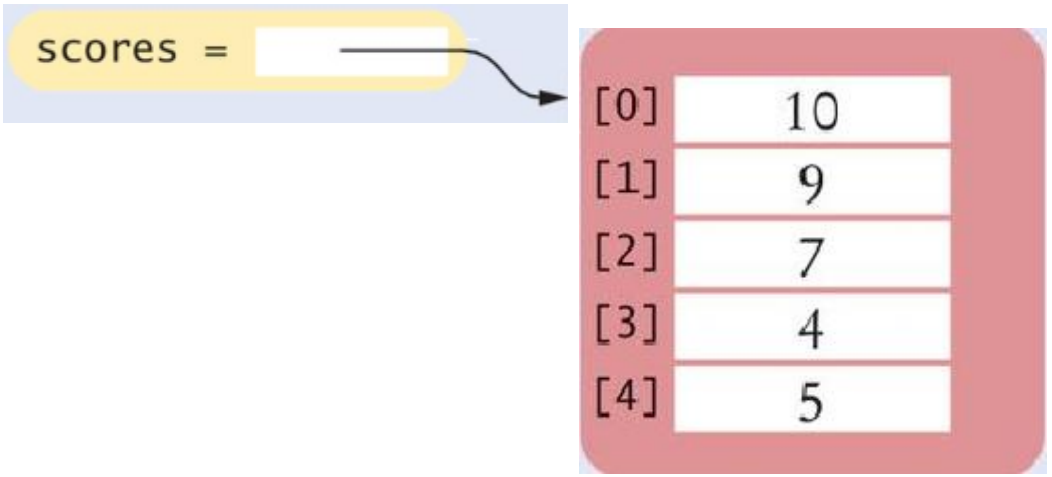
```
numbers = [100, 123]
numbers[1] = 5                 # list is now [100, 5]
```

List, List contents and Aliases

- A list variable contains a *reference* to the list contents. The *reference* is the memory address of the list contents. When copy a list to another, both refer the same.

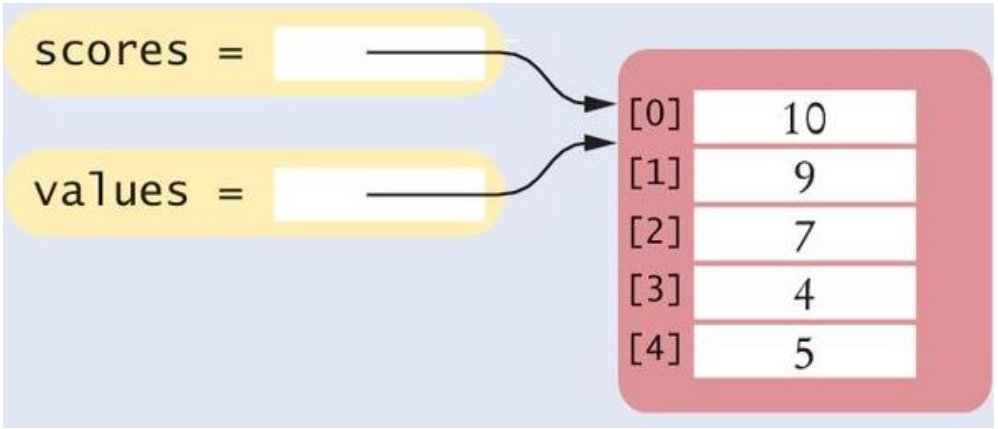
```
scores = [10,9,7,4,5]
```

List variable List content




```
Scores=[10,9,7,4,5]
values = scores
```

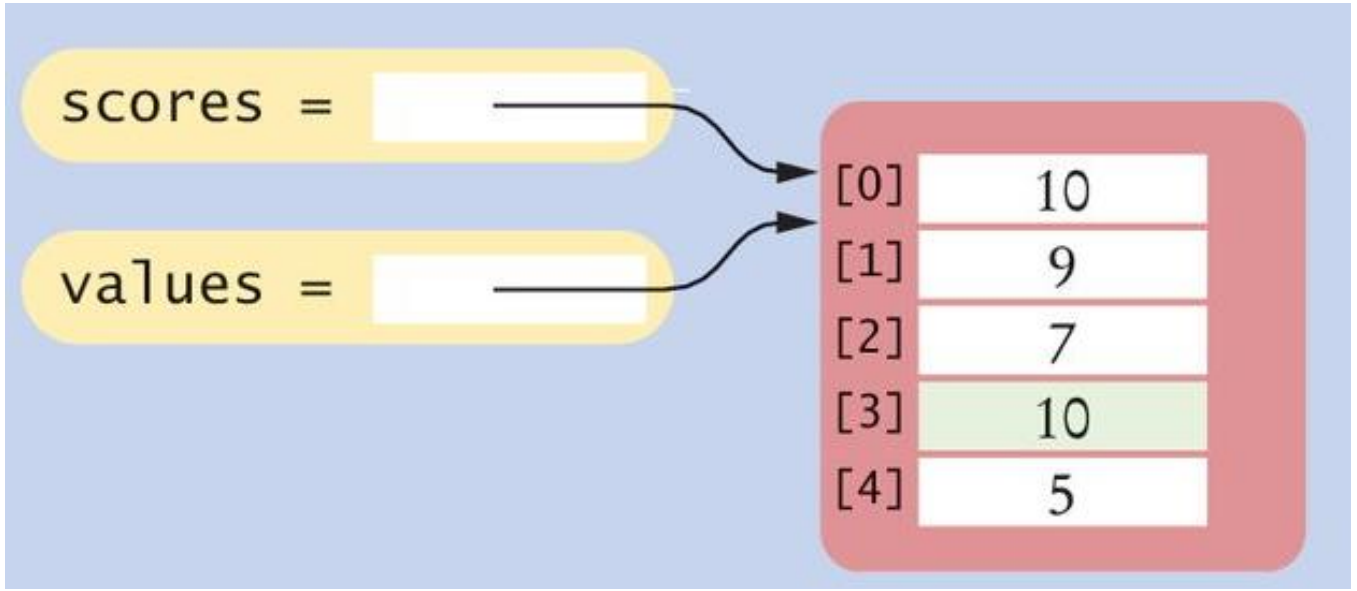
List variable List content



Modifying content

- As both lists refer to the same reference list, changing the content using any list variable can affect other.

```
Values[3]=10
```



List operations : append() and insert()

- append(item) allows to add a new item at the end of the list

```
scores = [10,9,7,4,5]
scores.append(25)
# new list 10,9,7,4,5,25
```

- insert(index,item) allows to add a item at a specific location of the list

```
scores = [10,9,7,4,5]
scores.insert(2,100)
# new list 10,9,100,7,4,5,25
```

List operations : finding an element

- To check whether an element is present in the list

```
scores = [10,9,7,4,5]
if 7 in scores:
   # do something
```

- Return the index of the element

```
scores = [10,9,7,4,5]
scores.index(7) #returns 2
```

List operations : Remove() and slicing

- pop(index) : Remove an element by specifying the index. Items after the specified index are moved up to fill the gap. Length decreases by 1

```
scores = [10,9,7,4,5]
scores.pop(1)
#new list 10,7,4,5
```

Slicing the list and store elements in another list

```
scores = [10,9,7,4,5]
newscores = scores[1:4]
#newscores 9,7,5
```

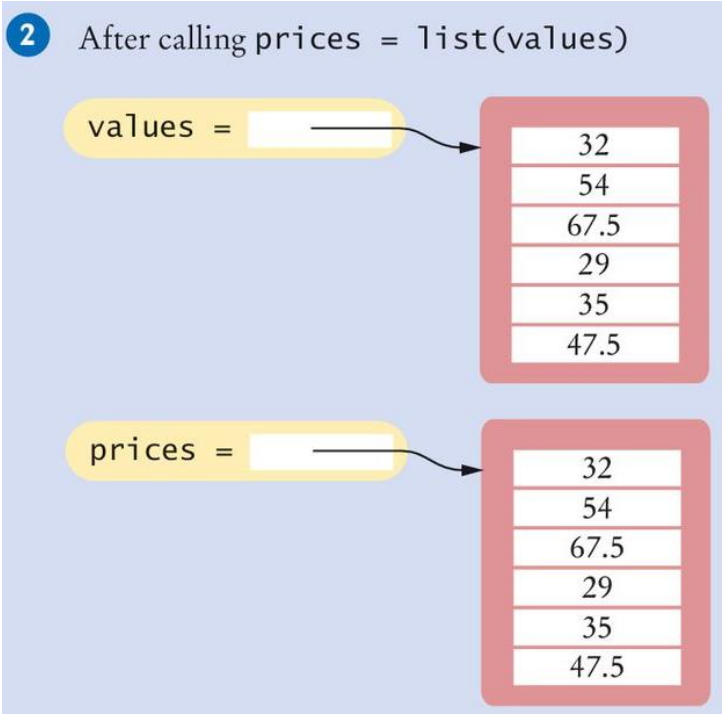
Concatenation and list()

- List concatenation is similar to the Strings

```
scores = [10,9,7,4,5]
newlist=scores+[45,78]
print(newlist) #10,9,7,4,5,45,78
```

List(list_name) to copy elements of a existing list

```
values=[32,54,67.25…..]
prices=list(values)
#this is not similar to prices=values
```



Tuples - Introduction

- Collection of ordered elements. Duplicates are allowed
- Main difference between list and tuples: elements cannot be modified/immutable
- Also you can cannot add, delete items like lists
- Tuple is a good fit if you do not want to change the values during the execution

```
test_tuple = (5, 10, 15,”hello”,45.7) #round brackets for tuples
print(test_tuple[2])
print(test_tuple[1:4])
print(10 in test_tuple)
newlist =list(test_tuple) # copy elements to a list
newtuple=test_tuple+(1,2,3)
```

Sets - Introduction

- Unordered and unindexed collection.
- You cannot modify the existing items once you added
- Duplicate members are not allowed

```
countries = {"UK","AUS","NZ"} #curly brackets
print(thisset)
for x in countries:
    print(x)  # AUS, NZ, UK
print("UK" in countries)
countries.add("US") #set(['US', 'AUS', 'NZ', 'UK'])
countries.update(["SL","IND","DE"]) # set(['DE', 'NZ', 'US', 'AUS', 'UK', 'SL', 'IND'])
countries.remove("IND") # remove raise an error if the item does not exist
countries.discard("IND")  # it does not raise an error
countries.pop() # as sets are unorded, not sure which item was deleted
print(countries) # set(['NZ', 'US', 'AUS', 'UK', 'SL'])
set2 = {"CAN", "PK", "ML"}
updated_set=countries.union(set2)
print(updated_set) # set(['ML', 'AUS', 'US', 'NZ', 'CAN', 'UK', 'SL', 'PK'])
```

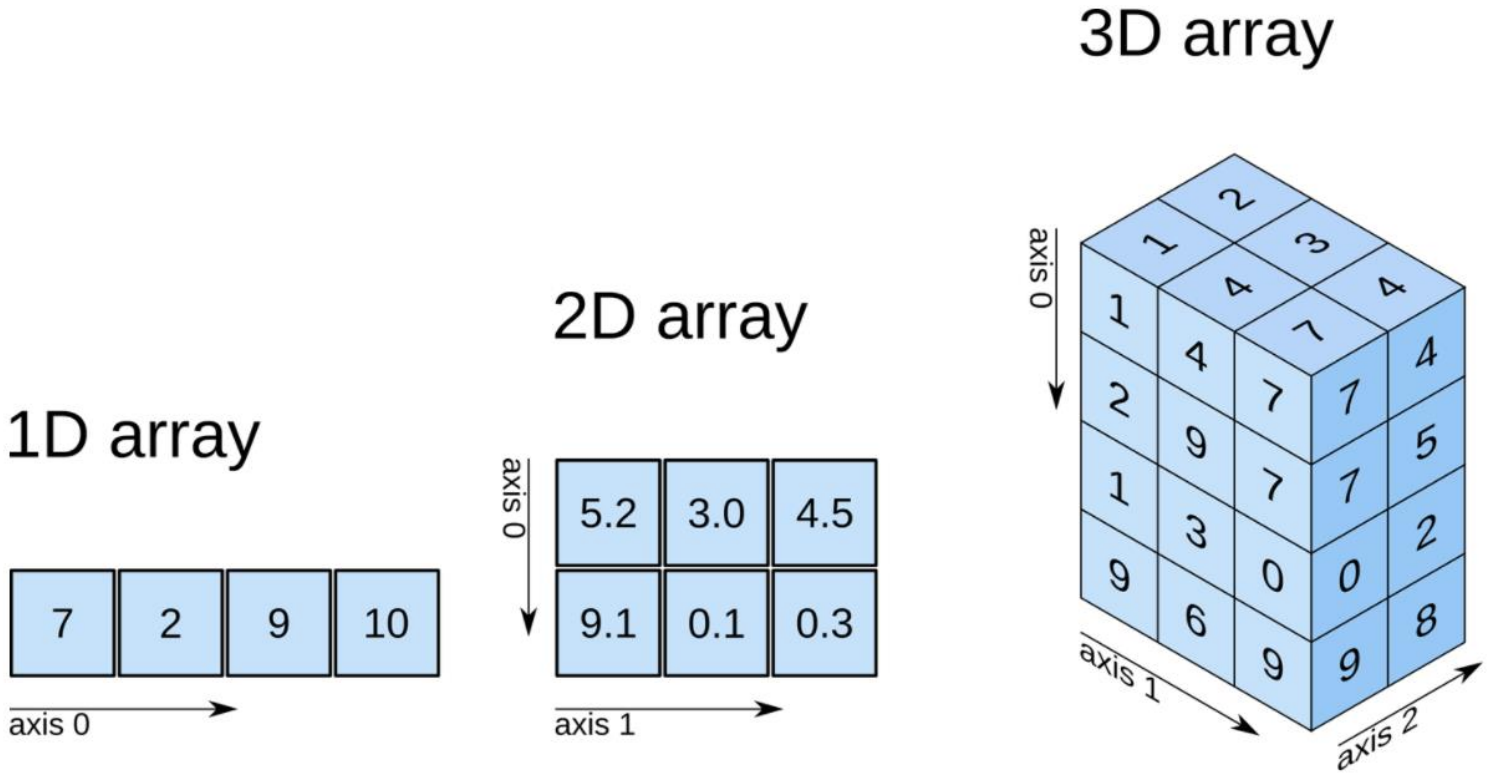
Dictionaries - Introduction

- Unordered, indexed and changeable collection.
- Each member inside a dictionary consist of “key” and a “value”

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict) #print whole dictionary
print(thisdict["model"]) #print the value of the model
thisdict["year"] = 2018 #update the value of year
for x in thisdict:
  print(x) #print keys
for x in thisdict.values():
  print(x) #print values
for x, y in thisdict.items():
  print(x, y) #print items
if "model" in thisdict:
    #see the key is there
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
thisdict["color"] = "red" #add a new key
thisdict.pop("model") # remove key - model
print(thisdict)
```

Challenge

- Check how to manage 2 dimensional, multidimensional arrays/lists in python



Summary

- Before choosing a collection for a specific task, the developer need to check several aspects (order of items, indexed ?, modifiable, duplication allowed?)
- Each collection type comes with set of predefined functions in order to add, remove, modify items

Collection	**ordered**	**indexed**	**modifiable**	**Duplication**
Array	yes	yes	yes	yes
List	yes	yes	yes	Yes
Tuple	yes	yes	no	yes
Set	no	no	no	no
Dictionary	no	yes	Yes(values only)	Key value pair