

# Programming 201

Introduction to Python Data Structures

**Kuldeep Singh Sidhu**

*kuldeepsinghsidhu.com*

# By the end of this session

- You will know What a data structure is!
- You will have an understanding of Primitive vs Non-Primitive data structures
- Will have done a lot of **hands-on** coding and using Non-Primitive data structures

str  
int/  
bool  
float

# Data Structure

Data structure is a **data organization, management, and storage format** that enables efficient access and modification

```
1. isEarth='True' # str
2. name='Earth' # str
3. rank=3 # int
4. radius=6371.0088 # float
5. aliensFound=False # bool
6. # 5 X 8 = 40 # SBT
7. print(f''' # name, DoB, Bal
8. {isEarth}: {type(isEarth)} # 1,000/000
9. {name}: {type(name)} + 3
10. {rank}: {type(rank)} => 000, 000
11. {radius}: {type(radius)} 
12. {aliensFound}: {type(aliensFound)} 
13. ''')
```

*This data is small and simple but in real world it might get complicated*

127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET / HTTP/1.1" 200 729 "-" "Mozilla/5.0  
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/blank.gif HTTP/1.1" 200 431 "h  
fox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/folder.gif HTTP/1.1" 200 509 "I  
efox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:11:37 +0530] "GET /icons/text.gif HTTP/1.1" 200 513 "ht  
ox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:11:38 +0530] "GET /favicon.ico HTTP/1.1" 404 500 "-" "M  
127.0.0.1 - - [31/Oct/2017:11:12:05 +0530] "GET /tecmint/ HTTP/1.1" 200 787 "http://l  
0"  
127.0.0.1 - - [31/Oct/2017:11:12:05 +0530] "GET /icons/back.gif HTTP/1.1" 200 499 "ht  
01 Firefox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /tecmint/Videos/ HTTP/1.1" 200 817 "h  
101 Firefox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /icons/compressed.gif HTTP/1.1" 200 1  
) Gecko/20100101 Firefox/56.0"  
127.0.0.1 - - [31/Oct/2017:11:13:58 +0530] "GET /icons/movie.gif HTTP/1.1" 200 527 "h  
0/20100101 Firefox/56.0"  
::1 - - [31/Oct/2017:11:26:57 +0530] "GET /ravi HTTP/1.1" 404 494 "-" "Mozilla/5.0 (X  
.36)"  
::1 - - [31/Oct/2017:11:26:57 +0530] "GET /favicon.ico HTTP/1.1" 404 500 "http://loca  
ome/60.0.3112.90 Safari/537.36"  
::1 - - [31/Oct/2017:11:27:20 +0530] "GET /anusha HTTP/1.1" 404 496 "-" "Mozilla/5.0  
37.36"  
Waiting for data... (interrupt to abort) -add - search - update,  
delete - s by 1

Source: request log: Real world data... KuldeepSinghSidhu.com 4

# Data Structures

Data structures allow us to efficiently

**store and manage**

data

# Types of data structures

# single value

# collection

Primitive	Non- Primitive
int	list
float	tuple
str	set
bool	dict
	any other user defined...

Primitive types store scalar/single values, while non-primitive store multiple values.  
Non primitive can be a collection of primitive and non primitive data types.

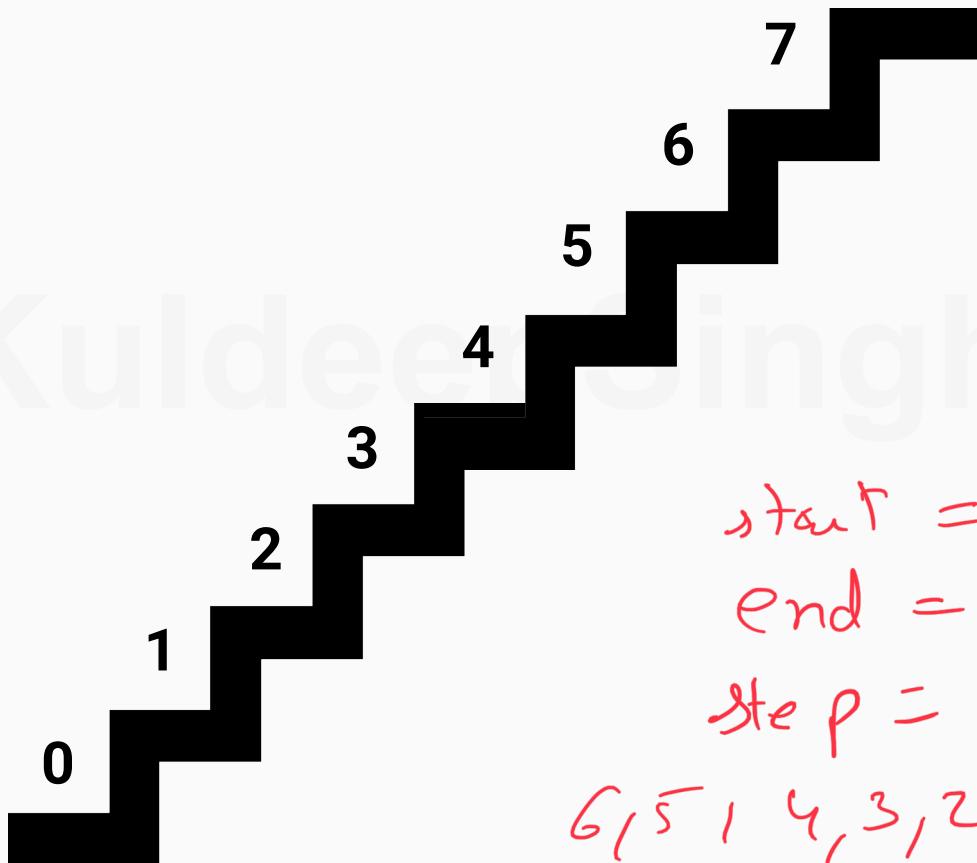
# list

```
1. # a list is represented by []
2.
3. ## empty list
4. products = []
5.
6. ## non-empty list
7. player = ['ronaldo', 7, 1.87, True]
8.
9. ## another list
10. grocery = [3, 'chairs', 3, 'tables', 1.5, 'apples', True, ['ronaldo', 7, 1.87, True]]
11.
12. print(player, '\n', grocery)
13. print(type(player), '\n', type(grocery))
14.
15. ### NOTES ###
16. # 1: A list can contain any data type (including a list)
17. # 2: A list can contain duplicates
```

## list

```
1. # indexing in a list
2. # list is 0-indexed
3. #[0 1 2 3..... -3 -2 -1]
4.
5. avengers = ['thor', 'iron-man', 'hulk', 'cap']
6. ##### 0           1           2           3
7. ##### -4          -3          -2          -1
8.
9. print(avengers[0], avengers[-4])
10. print(avengers[1], avengers[-3])
```

list



start = 0  
end = 1  
step = -1

6, 5, 4, 3, 2, 1

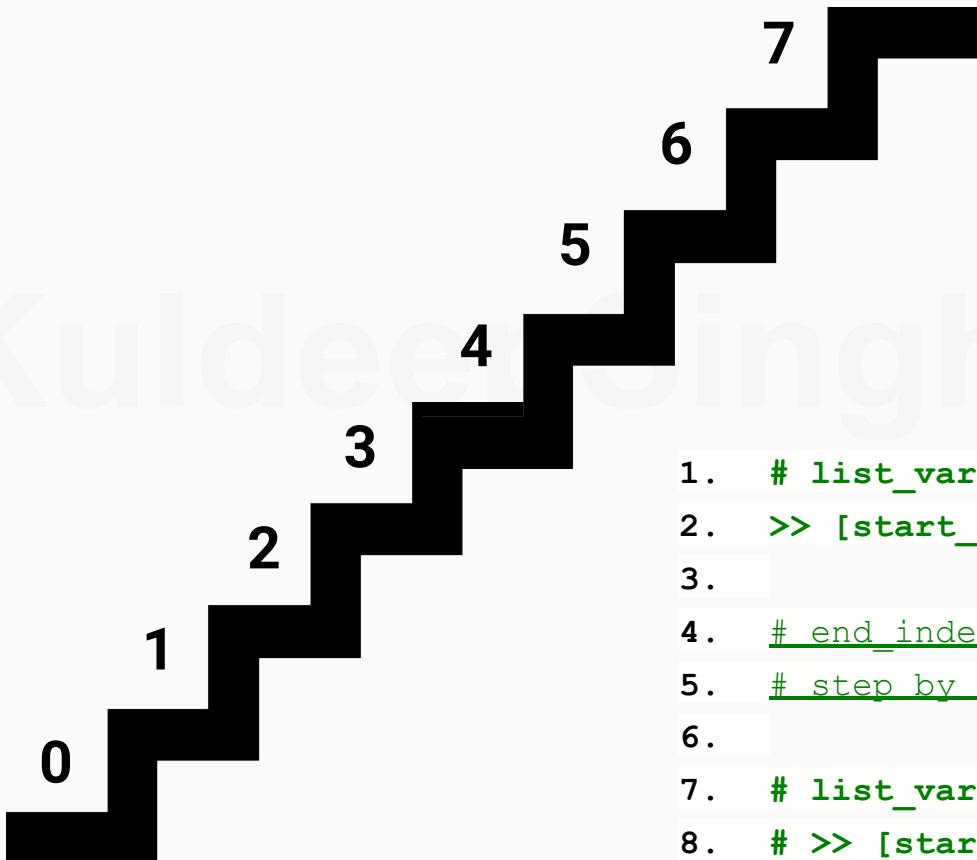
start = 0  
end = 5  
0, 1, 2, 3, 4, 5  
assumed by default  
step = 1

start = 1  
end = absolute end  
step = 2  
1, 3, 5, 7

# list

```
1. # slicing a list >>> my_list[from:till:step]
2.
3. avengers = ['thor', 'iron-man', 'hulk', 'cap']
4.
5. print(avengers[1:3]) # slicing
6. print(avengers[:3]) # slicing one side
7. print(avengers[1:]) # slicing one side
8. print(avengers[0:3:2]) # steps
9. print(avengers[::-1]) # reverse steps
10.
11. ### NOTES ###
12. # 1: my_list[from:till:step] by default from=0, till=list length, step=1
13. # 2: from is included, till is not
```

# list



```
1. # list_var[start_index:end_index]
2. >> [start_index, start_index+1...end_index-1]
3.
4. # end_index is NOT included
5. # step by default is 1
6.
7. # list_var[start_index:end_index: step]
8. # >> [start_index, start_index+step...end_index-1] 1
```

- ```
1. # first 10 english alphabets
2. alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
3. print(alphabets[0]) - A
4. print(alphabets[-3]) - H
5. print(alphabets[5]) - F
6. print(alphabets[1]) - B
7. print(alphabets[-0]) - A ← -0 == 0
8. print(alphabets[1:5]) [B, C, D, E] start = 1 end = 5
9. print(alphabets[:1]) [A]
10. print(alphabets[5:]) [F, G, H, I, J]
11. print(alphabets[1:5:2]) [B, D] - start = 1 end = 5 step = 2
12. print(alphabets[::-1])
    = [J, I, H, G, F, E, D, C, B, A]
```

## list

```
1. ## step by step understanding
2.
3. # how to access each element 1 by 1 (traversing)
4. # using a loop
5.
6. avengers = ['thor', 'iron-man', 'hulk', 'cap']
7.
8. for hero in avengers:
9.     print(hero)
10.
11. ### NOTES ###
12. # 1: here hero is a temp variable whose value changes iteratively
13. # 2: it starts and ends automatically
```

# list

```
1. # manipulating/updating the list elements
2.
3. avengers = [ 'thor', 'iron-man', 'hulk', 'cap' ]
4.
5. print(avengers, avengers[ 0 ])
6. avengers[ 0 ] = 'god of thunder'
7. print(avengers, avengers[ 0 ])
8.
9. print(avengers, avengers[ 1 ], type(avengers[ 1 ]))
10. avengers[ 1 ] = False
11. print(avengers, avengers[ 1 ], type(avengers[ 1 ]))
12.
13. ### NOTES ###
14. # 1: list is a mutable data structure, that is it is editable
15. # 2: you can change an element of the list from one data type to another
```

## list

```
1. # adding elements to the list using append(element) and insert(pos, element)
2.
3. avengers = ['thor', 'iron-man', 'hulk', 'cap']
4.
5. print(avengers)
6. avengers.append('gamora')
7. print(avengers)
8. avengers.append('nebula')
9. print(avengers)
10.
11. avengers.insert(1, 'loki')
12. print(avengers)
13.
14.
15. ### NOTES ###
16. # 1: append() is used to add elements to a list (at the end)
17. # 2: insert() is used to insert element at a specific location in a list
```

## list

```
1. # merging list into a list using extend()
2. avengers = ['thor', 'iron-man', 'hulk', 'cap']
3. DC = ['batman', 'superman', 'aquaman']
4.
5. print(avengers)
6. avengers.extend(DC)
7.
8. print(avengers)
9.
10. # here DC is untouched
11. print(DC)
12.
13. ### NOTES ###
14. # 1: extend() is used to merge one list to another
15. # 2: the list is added at the end
```

1. ## What is the difference between append vs insert vs extend?
- 2.
3. # append() adds element at the end of the list
4. # insert() adds element at desired index
5. # extend() merges a list into an existing list

# list

```
1. # removing elements from a list
2. avengers = ['thor', 'iron-man', 'hulk', 'cap']
3. avengers.insert(1,'loki')
4. avengers.append('thanos')
5. print(avengers)
6. # now avengers is ['thor', 'loki', 'iron-man', 'hulk', 'cap', 'thanos']
7. # remove()
8. avengers.remove('thanos')
9. print(avengers)
10. # del
11. del avengers[1]
12. print(avengers)
13. del avengers[::-2]
14. print(avengers)
15.
16. ### NOTES ##
17. # 1: we can use remove() to remove an element using the value(first occurrence)
18. # 2: we can use del to remove an element using the index, del here is a keyword
19. # 3: we can use del to remove list slices
```

# list

```
1. # removing element from the list using pop() >>> returns the deleted
   element
2. avengers = ['thor', 'iron-man', 'hulk', 'cap']
3. lastHero = avengers.pop()
4. print(lastHero, '\n', avengers)
5. secondHero = avengers.pop(1)
6. print(secondHero, '\n', avengers)
7.
8. ### NOTES ###
9. >>> Differences
10. # 1: remove(): removes based on value
11. # 2: del: deletes element or slices using index
12. # 3: pop(): removes an element and returns it
```

## list

```
1. # sorting the list
2. avengers = ['thor', 'iron-man', 'hulk', 'cap']
3.
4. avengers.sort()
5.
6. print(avengers)
```

## list

```
1. # searching the list (Quick Search)
2. ## using in keyword to quickly check if any element is present in the list
   or not
3. avengers = ['thor', 'iron-man', 'hulk', 'cap']
4. print('groot' in avengers)
5. print('iron-man' in avengers)
```

## list

```
1. ## doing element by element comparison
2. avengers = ['thor', 'iron-man', 'hulk', 'cap']
3.
4. for myHero in avengers:
5.     if myHero=='hulk':
6.         print('found hulk')
7.         break
```

```
1. # count the number of 5 rs coins in the list
2. # when does first 5 rs coin appear
3.
4. coins = [1,2,5,6,5,7,8,5,10]
5. count=0
6. locFound=False
7. loc=0
8. for coin in coins:
9.     if coin==5:
10.         count+=1
11.         locFound=True
12.     if locFound==False:
13.         loc+=1
14.
15. print(f'Number of 5 rupees coin found are {count}')
16. print(f'First 5 rs coin appears at {loc}')
17.
18. ## directly use count to count an element
19. print(f'Number of 5 rupees coin found are {coins.count(5)}')
20. ## directly use index to find index of an element
21. print(f'First 5 rs coin appears at {coins.index(5)})
```

## Manage list data:

- Add
- Delete
- update
- Search
- sort
- Go 1 by 1

## tuple

```
1. # Tuples are created using ()
2.
3. ## Tuples are popular for their read only nature
4. ## that is tuples are immutable >> no add/update/delete/sort
5.
6. ### Empty tuple
7. empty = ()
8.
9. ### filled tuple
10. days = ('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')
11. print(days, type(days))
12. #### reading
13. print(days[0])
14. #### searching
15. print('mon' in days)
16.
17. # days[0]='funday' # ERROR: cannot update tuple
```

## When do we use tuples:

When we don't want data to be modified by anyone for system stability

Eg:

For a ecommerce website,

Model numbers, product id will be stored in a tuple

While things like sale price, discount, qty available will be stored in a list

# Packing and Unpacking of elements

```
1. # unpacking elements
2.
3. #tuple
4. flavours= ('sweet', 'sour') #packed
5. f1,f2 = flavours
6. print(f1,f2)
7.
8. # list
9. flavours= ['sweet', 'sour'] #packed
10. f1,f2 = flavours
11. print(f1,f2)
12.
13. ## we generally do it for upto 3 to 5 elements
14. ## the number of elements packed to unpacked should be equal
15. ### errors
16. #### f1,f2,f3 = ['sweet', 'sour']
17. #### f1,f2 = ['sweet', 'sour', 'salty']
```

## sets

```
1. # sets in python are represented by {}
2.
3. # sets allow us to store distinct/unique elements
4. colors = {'red','cyan','cyan','orange','green','red','Red'}
5. print(colors)
6. # traversing
7. for c in colors:
8.     print(c)
9. # sets are unordered so no indexing is supported or sorting
10. # print(colors[1])
```

## sets

```
1. # sets is same as sets in math class
2.
3. setA ={1,2,3,4,5}
4. setB ={4,5,6,7,8}
5.
6. # union
7. print(setA.union(setB))
8. # intersection
9. print(setA.intersection(setB))
10. # difference
11. print(setA.difference(setB)) # elements of a that are not in b
```

## sets

```
1. # empty set
2. emptySet = set()
3. print(emptySet, type(emptySet))
4.
5. # add things to set
6. emptySet.add(1)
7. emptySet.add(2)
8. emptySet.add(3)
9. print(emptySet)
10.
11. # can add duplicates as many times we want but will only be stored once
12. emptySet.add(1)
13. emptySet.add(1)
14. print(emptySet)
15.
16. # remove
17. emptySet.remove(2)
18. # emptySet.remove(2) # ERROR: Cannot remove something that is not present
19. print(emptySet)
```

# len()

```
1. # len
2.
3. # str
4. name = 'ronaldo'
5.
6. # list
7. player = ['ronaldo', 7, 1.87, True]
8. ## another list
9. grocery = [3, 'chairs', 3, 'tables', 1.5, 'apples', True, ['ronaldo', 7, 1.87, True]]
10.
11. # tuple
12. days = ('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')
13.
14. # sets
15. colors = {'red', 'cyan', 'cyan', 'orange', 'green', 'red', 'Red'}
16.
17. print(len(name), len(player), len(grocery), len(days), len(colors))
```

# dict

```
1. # python dictionaries are a special data structure
2. # key:value pair (data)
3.
4. # example an oxford dictionary word: meaning
5. # you search the word which is the key
6. # to get the meaning which is the value
7.
8. oxford = {'apple':'good fruit', 'trump':'president', 'google':'company'}
9. print(oxford['google'])
```

To be discussed in detail...



<https://www.hackerrank.com/domains/python>

# Practice

This is just the beginning