Sequence data type

• **Strings** - Sequence of characters - " (or) ' • **Tuples** - Sequence of compound data - () • **Lists** - Sequence of multi-data type objects - [] •** Arrays** - Sequence of constrained list of objects (all objects of same datatype)

`- using array module from array package`

• **Dictionary**- Sequence of key-value pairs - {} •**Sets** - Sequence of unordered collection of unique data • **Range** - Used for looping - using built-in range( )

• These can offer unique functionalities for the variables to contain and handle more than one data datatype at a time • Supports operations such as indexing, slicing, concatenation, multiplication etc.,

STRINGS

```
strsample  = 'learning '

strsample[8]
```

```
' '
```

```
lstNumbers = [1, 2, 3, 3, 3, 4, 5]# List with numbers
# (hav ing dupL i cate vaL ues)


print(lstNumbers)
```

```
[1, 2, 3, 3, 3, 4, 5]
```

```
lstSample   = [ 1, 2, ' a ' , ' saw ' , 2,strsample ]# List with mixed data types
# (having numbers and stings)

print(lstSample)
```

```
[1, 2, ' a ', ' saw ', 2, 'learning']
```

```
from array import *
arrsample = array('i',[1,2,3,4])# Array with numbers
for x in arrsample: print(x)
```

```
1
2
3
4
```

```
tupsample    = (1, 2, 3, 4, 3, ' py' )  # tup L e
```

```
tuplesample = (1, 2, 'sample')
print(tuplesample)
tuplesample[2]
```

```
(1, 2, 'sample')
'sample'
```

```
dictsample = {1:'first',
              'second' :2,
              3:3,
              'four':'4'}
dictsample['second']
```

```
2
```

```
dict_list = dict([('first', 1), ('second', 2), ('four', 4)])
type(dict_list)
```

```
dict
```

```
setSample   = {'example ' , 24, 87.5, 'data' , 24, 'data' } # set
setSample
# Unordered Data Collection
```

```
{24, 87.5, 'data', 'example '}
```

```
rangesample= range(1,12,4)
print(rangesample)

for x in rangesample: print(x)
```

```
range(1, 12, 4)
1
5
9
```

Sequence data operations: Indexing

Indexing just means accessing elements.

To access elements, the square brackets can be used. There are many methods to access elements in python.

index() method finds the first occurrence of the specified value and returns its position

`Syntax: object.index(sub[, start[, end]] ), object[index]`

• Index of the element is used to access an element from ordered sequences

• The index starts from 0

• Negative indexing is used to access elements from the end of a list

• In negative indexing, the last element of a list has the index -1

String: Indexing

```
sample_str = 'learning' #String
```

```
sample_str.index("arn")
```

```
2
```

```
sample_str.index("z")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-6d5cace1c9d0> in <cell line: 1>()
----> 1 sample_str.index("z")

ValueError: substring not found
```

```
sample_str.index("nin")
```

```
4
```

```
sample_str[4] #to find out what character is at index 4
```

```
'n'
```

```
# Negetive Indexing
sample_str[-1]
```

```
'g'
```

```
#Negetive Index out of range
sample_str[-10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-19-9c1b16da002e> in <cell line: 2>()
      1 #Negetive Index out of range
----> 2 sample_str[-10]

IndexError: string index out of range
```

List: Indexing

Syntax: list_name.index(element, start, end)

```
sample_lst = [ 1, 2, ' a ' , 'saw' , 2] #List
```

```
# List indexing using index method
sample_lst.index("saw")
```

```
3
```

```
#List indexing With Corresponding Position
sample_lst[3]
```

```
'saw'
```

```
#Negative indexing
sample_lst[-1]
```

```
2
```

Array_Indexing

```
from array import *
arrsample = array('i',[1,2,3,4]) # array with integer type
```

```
#printing the values of array sample
for x in arrsample: print(x)
```

```
1
2
3
4
```

```
#Negetive indexing in array sample
arrsample[-1]
```

```
4
```

Tuple_Indexing

```
tupsample = (1, 2, 3, 4, 3, "py" )
```

```
#Using index to find index value of py
tupsample.index("py")
```

```
5
```

```
#using position of the element
tupsample[3]
```

```
4
```

Set_Indexing

```
set_sample = {1, 2, 3, 4, 3, "py" } # Sets
set_sample
```

```
{1, 2, 3, 4, 'py'}
```

```
#indexing in sets
set_sample[3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-f42e6637e88d> in <cell line: 2>()
      1 #indexing in sets
----> 2 set_sample[3]

TypeError: 'set' object is not subscriptable
```

Dictionary_Indexing

```
dictsample = {1:'first', 'second' :2, 3:3, 'four':'4'} # dictionary
dictsample
```

```
{1: 'first', 'second': 2, 3: 3, 'four': '4'}
```

```
# dictsample[2] # Indexing Values not applicable in Dictionary
dictsample.keys()
```

```
dict_keys([1, 'second', 3, 'four'])
```

```
dictsample[1]
# it works to find the correspoinding values called as key value pairs
```

```
'first'
```

Sequence data operations: Slicing

• The slice() constructor creates a slice object representing the set of indices specified by range(start, stop, step)

• `Syntax: slice(stop), slice(start, stop, step)`

• If a single parameter is passed, start and step are set to None

```
#String Slicing
strsample[slice(4)] # we will get substring from index 0 to 4 exlcuding 4
```

```
'lear'
```

```
strsample[slice(1,6,2)] # Start stop step (3S)
```

```
'eri'
```

```
strsample[:]
```

```
'learning'
```

```
strsample[::2]
```

```
'lann'
```

```
strsample[::-1]
```

```
'gninrael'
```

```
#Similarly applicable for list and tuple as well
lstSample[slice(3)]
```

```
[1, 2, ' a ']
```

```
lstSample[:4]
```

```
[1, 2, ' a ', ' saw ']
```

slicing is not possible in dictionaries and sets

```
Start coding or generate with AI.
```