Cardamom Planters' Association College

(Re-accredited with 'B' Grade by NAAC)
Pankajam Nagar, Bodinayakanur – 625513

Department CS & IT

TOPIC NAME: PYTHON – LIST, TUPLES, DICTIONARY

(2024-2025)

Prepared by

S.Rohini
Assistant Professor
Department of CS & IT
CPA College
Bodinayakanur

1. ELEMENTARY DATA ITEMS

In python, elementary data items (also known as primitive data types) are the basic building blocks of data. These include:

Integer: Whole numbers, positive or negative, without decimals.

Floating-point numbers (float): Numbers that contain a decimal point.

Strings (str): A sequence of characters enclosed in single or double quotes.

Booleans (bool): Represents one of two values: True or False.

Complex(complex): Number with real and imaginary parts.

Program:

```
a = 5
print("Type of a: ", type(a))
b = 5.0
print("\nType of b: ", type(b))
c = 2 + 4j
print("\nType of c: ", type(c))
```

Output:

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```

Program:

```
str = 'Computer Science'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:7])     # Prints characters starting from 3rd to 7th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

Output

```
Computer Science
C
mpute
mputer Science
Computer ScienceComputer Science
Computer ScienceTEST
```

Program:

```
x = bool(5)
y=bool(0)
```

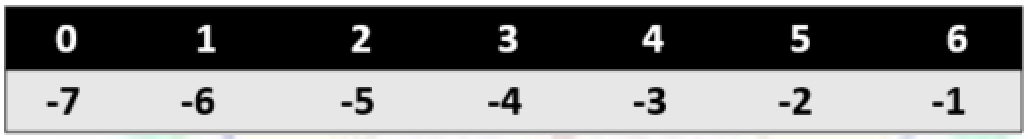
```
#display x and y:
print(x)
print(y)
#display the data type of x and y:
print(type(x))
print(type(y))
```

Output:

```
True
False
<class 'bool'>
<class 'bool'>
```

2. LIST

To store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index. The index value starts from 0 and goes on until the last element called the **positive index**. There is also **negative indexing** which starts from -1 enabling you to access elements from the last to first.



0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Adding Elements
Adding the elements in the list can be achieved using the append(), extend() and insert() functions.

- The append() function adds all the elements passed to it as a single element.
- The extend() function adds the elements one-by-one into the list.
- The insert() function adds the element passed to the index value and increase the size of the list too.

Deleting Elements

- To delete elements, use the del keyword which is built-in into Python but this does not return anything back to us.
- If you want the element back, you use the pop() function which takes the index value.
- To remove an element by its value, you use the remove() function.

Accessing Elements
Accessing elements is the same as accessing Strings in Python. You pass the index values and hence can obtain the values as needed.

Other Functions
You have several other functions that can be used when working with lists.

- The len() function returns to us the length of the list.

- The index() function finds the index value of value passed where it has been encountered the first time.
- The count() function finds the count of the value passed to it.
- The sorted() and sort() functions do the same thing, that is to sort the values of the list. The sorted() has a return type whereas the sort() modifies the original list.

Sample Program

```
m= [1, 2, 3, 'example', 3.132] #creating list with data
print(m)
m= [1, 2, 3]
print(m)
print('\n adding element')
m.append([555, 12]) #add as a single element
print(m)
m.extend([234, 'more_example']) #add as different elements
print(m)
m.insert(1, 'insert_example') #add element i
print(m)
m= [1, 2, 3, 'example', 3.132, 10, 30]
print('\n deleting element')
del m[5] #delete element at index 5
print(m)
m.remove('example') #remove element with value
print(m)
a = m.pop(1) #pop element from list
print('Popped Element: ', a, ' List remaining: ', m)
m.clear() #empty the list
print(m)
print('\n accessing element')
a= [1, 2, 3, 'example', 3.132, 10, 30]
for element in a: #access elements one by one
    print(element)
print(a) #access all elements
print(a[3]) #access index 3 element
print(a[0:2]) #access elements from 0 to 1 and exclude 2
print(a[::-1]) #access elements in reverse
print('\n other functions')
b= [1, 2, 3, 10, 30, 10]
print(len(b)) #find length of list
print(b.index(10)) #find index of element that occurs first
print(b.count(10)) #find count of the element
print(sorted(b)) #print sorted list but not change original
b.sort(reverse=True) #sort original list
print(b)
```

Output

```
[1, 2, 3, 'example', 3.132]
[1, 2, 3]
```

adding element

```
[1, 2, 3, [555, 12]]
[1, 2, 3, [555, 12], 234, 'more_example']
[1, 'insert_example', 2, 3, [555, 12], 234, 'more_example']
```

deleting element

```
[1, 2, 3, 'example', 3.132, 30]
[1, 2, 3, 3.132, 30]
Popped Element:  2  List remaining:  [1, 3, 3.132, 30]
[]
```

accessing element

```
1
2
3
example
3.132
10
30
[1, 2, 3, 'example', 3.132, 10, 30]
example
[1, 2]
[30, 10, 3.132, 'example', 3, 2, 1]
```

other functions

```
6
3
2
[1, 2, 3, 10, 10, 30]
[30, 10, 10, 3, 2, 1]
```

3. DICTIONARY

To store **key-value** pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added. Now the constant values here are Name and the Phone Numbers which are called as the keys. And the various names and phone numbers are the values that have been fed to the keys. If you access the values of the keys, you will obtain all the names and phone numbers. So that is what a key-value pair is. And in Python, this structure is stored using Dictionaries.

Creating a Dictionary

Dictionaries can be created using the flower braces or using the dict() function. You need to add the key-value pairs whenever you work with dictionaries.

Changing and Adding key, value pairs

To change the values of the dictionary, you need to do that using the keys. So, you firstly access the key and then change the value accordingly. To add values, you simply just add another key-value pair

Deleting key, value pairs

- To delete the values, you use the pop() function which returns the value that has been deleted.
- To retrieve the key-value pair, you use the popitem() function which returns a tuple of the key and value.
- To clear the entire dictionary, you use the clear() function.

Accessing Elements

You can access elements using the keys only. You can use either the get() function or just pass the key values and you will be retrieving the values.

Other Functions

You have different functions which return to us the keys or the values of the key-value pair accordingly to the keys(), values(), items() functions accordingly.

Sample Program

```
print("creating dictionary with elements")
s= {1: 'Python', 2: 'Java'}
print(s)
s= {'First': 'Python', 'Second': 'Java'}
print(s)
print("\n changing element")
s['Second'] = 'C++'
print(s)
print("\n adding key-value pair")
s['Third'] = 'Ruby'
print(s)
print("\n pop element")
s= {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
a = s.pop('Third')
print('Value:', a)
print('Dictionary:',s)
print("\n pop the key value pair")
b = s.popitem()
print('Key, value pair:', b)
print(s)
print("\n empty dictionary")
```

```
s.clear()
print('n', s)
print("\n access element using keys")
s= {'First': 'Python', 'Second': 'Java'}
print(s['First'])
print(s.get('Second'))
s = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
print(s.keys()) #get keys
print(s.values()) #get values
print(s.items()) #get key-value pairs
print(s.get('First'))
```

Output

```
creating dictionary with elements
{1: 'Python', 2: 'Java'}
{'First': 'Python', 'Second': 'Java'}

changing element
{'First': 'Python', 'Second': 'C++'}

adding key-value pair
{'First': 'Python', 'Second': 'C++', 'Third': 'Ruby'}

pop element
Value: Ruby
Dictionary: {'First': 'Python', 'Second': 'Java'}

pop the key value pair
Key, value pair: ('Second', 'Java')
{'First': 'Python'}

empty dictionary
n {}

access element using keys
Python
Java
dict_keys(['First', 'Second', 'Third'])
dict_values(['Python', 'Java', 'Ruby'])
dict_items([('First', 'Python'), ('Second', 'Java'), ('Third', 'Ruby')])
Python
```

4. TUPLES

Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed.

Creating a Tuple
You create a tuple using parenthesis or using the tuple() function.
Accessing Elements
Accessing elements is the same as it is for accessing values in lists.
Appending Elements
To append the values, you use the ‘+’ operator which will take another tuple to be appended to it.
Other Functions
These functions are the same as they are for lists.

Sample Program

```
t= (1, 2, 3) #create tuple
print(t)
print("\naccess elements")
t= (1, 2, 3, 'edureka')
for x in t:
print(x)
print(t)
print(t[0])
print(t[:])
print(t[3][4])
t= (1, 2, 3)
print("\nadd element")
t= t + (4, 5, 6)
print(t)
t= (1, 2, 3, ['hindi', 'python'])
t[3][0] = 'english' #accesses the first element of that fourth inner list.
print(t)
print(t.count(2))
print(t.index(['english', 'python']))
```

Output:

```
(1, 2, 3)

access elements
1
2
3
edureka
(1, 2, 3, 'edureka')
1
(1, 2, 3, 'edureka')
e
```

```
add element
(1, 2, 3, 4, 5, 6)
(1, 2, 3, ['english', 'python'])
1
3
```