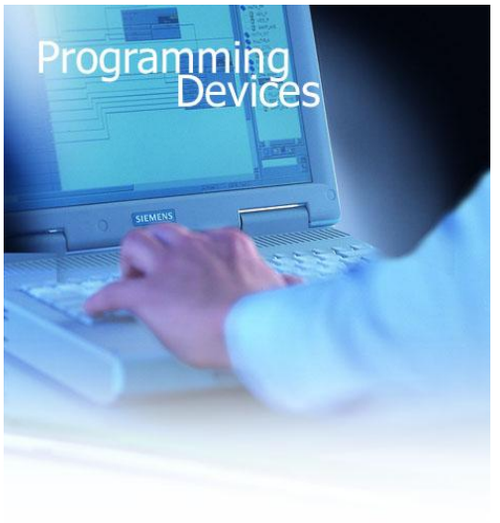
# UNIT 4

# Programming with Python



UNIT 4

# Chapter 1

# Strings

***After studying this lesson, students will be able to:***

- *Learn how Python inputs strings*
- *Understand how Python stores and uses strings*
- *Perform slicing operations on strings*
- *Traverse strings with a loop*
- *Compare strings and substrings*
- *Understand the concept of immutable strings*
- *Understanding string functions.*
- *Understanding string constants*

## Introduction

In python, consecutive sequence of characters is known as a string. An individual character in a string is accessed using a subscript (index). The subscript should always be an integer (positive or negative). A subscript starts from 0.

**Example**

```
# Declaring a string in python
>>>myfirst="Save Earth"
>>>print myfirst
Save Earth
```

**Let's play with subscripts**

To access the **first character** of the string

```
>>>print myfirst[0]
S
```

To access the **fourth character** of the string

```
>>>print myfirst[3]
e
```

To access the **last character** of the string

```
>>>print myfirst[-1]
>>h
```

To access the **third last character** of the string

```
>>>print myfirst[-3]
r
```

Consider the given figure

| **String A** | H | E | L | L | O |
|---|---|---|---|---|---|
| **Positive Index** | 0 | 1 | 2 | 3 | 4 |
| **Negative Index** | -5 | -4 | -3 | -2 | -1 |

Important points about accessing elements in the strings using subscripts

- Positive subscript helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.

  Example : A[0] or A[-5] will display 'H'
- Subscript 1 or –ve (n-1) displays the second element.

**Note:** Python does not support character data type. A string of size 1 can be treated as characters.

## Creating and initializing strings

A literal/constant value to a string can be assigned using a single quotes, double quotes or triple quotes.

### ✡ Enclosing the string in single quotes

**Example**

>>>print ('A friend in need is a friend indeed')

A friend in need is a friend indeed

**Example**

>>>print(' This book belongs to Raghav**\'s** sister')

This book belongs to Raghav's sister

As shown in example 2, to include the single quote within the string it should be preceded by a backslash.

### ✡ Enclosing the string in double quotes

**Example**

>>>print("A room without books is like a body without a soul.")

A room without books is like a body without a soul.

### ✡ Enclosing the string in triple quote

**Example**

>>>life="""\" Live as if you were to die tomorrow.

Learn as if you were to live forever.\"

---- Mahatma Gandhi """

>>> print life

" Live as if you were to die tomorrow.

Learn as if you were to live forever."

---- Mahatma Gandhi """

Triple quotes are used when the text is multiline.

In the above example, backslash (\) is used as an escape sequence. An escape sequences is nothing but a special character that has a specific function. As shown above, backslash (\) is used to escape the double quote.

| Escape sequence | Meaning | Example |
|---|---|---|
| \n | New line | >>> print "Hot\nCold"<br>Hot<br>Cold |
| | Tab space | >>>print "Hot\tCold"<br>Hot<br>Cold |

### By invoking raw_input() method

Let's understand the working of raw input() function

**Example**

```
>>>raw_input()
Right to education
'Right to education'
```

As soon as the interpreter encounters raw_input method, it waits for the user to key in the input from a standard input device (keyboard) and press Enter key. The input is converted to a string and displayed on the screen.

**Note:** raw_input( ) method has been already discussed in previous chapter in detail.

### By invoking input() method

**Example**

```
>>>str=input("Enter the string")
Enter the string hello
NameError: name 'hello' is not defined
```

Python interpreter was not able associate appropriate data type with the entered data. So a NameError is shown. The error can be rectified by enclosing the given input i.e. hello in quotes as shown below

| | |
|---|---|
| >>>str=input("Enter the String")<br>Enter the String "hello"<br>>>> print str<br>Hello | >>>str=input("Enter the String")<br>Enter the String'hello'<br>>>> print str<br>hello |

## Strings are immutable

Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example.

**Example**

```
>>>str='honesty'
>>>str[2]='p'
TypeError: 'str' object does not support item assignment
```

Python does not allowthe programmer to change a character in a string. As shown in the above example, str has the value 'honesty'. An attempt to replace 'n' in the string by 'p' displays a TypeError.

## Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

| String traversal using for loop | String traversal using while loop |
|---|---|
| A='Welcome'<br>>>>for i in A:<br>print i<br>W | A='Welcome'<br>>>>i=0<br>>>>while i<len(A)<br>print A[i] |

| | |
|---|---|
| e<br>l<br>c<br>o<br>m<br>e | i=i+1<br>W<br>e<br>l<br>c<br>o<br>m<br>e |
| A is assigned a string literal 'Welcome'.<br>On execution of the for loop, the characters in the string are printed till the end of the string is not reached. | A is assigned a string literal 'Welcome'<br>i is assigned value 0<br>The len() function calculates the length of the string. On entering the while loop, the interpreter checks the condition. If the condition is true, it enters the loop. The first character in the string is displayed. The value i is incremented by 1. The loop continues till value i is less than len-1. The loop finishes as soon as the value of I becomes equal to len-1, the loop |

## Strings Operations

| Operator | Description | Example |
|---|---|---|
| + (Concatenation) | The + operator joins the text on both sides of the operator | >>> 'Save'+'Earth'<br>'Save Earth'<br>To give a white space between the two words, insert a space before the closing single quote of the first literal. |
| * (Repetition ) | The * operator repeats the | >>>3*'Save Earth ' |

| | | |
|---|---|---|
| | string on the left hand side times the value on right hand side. | 'Save Earth Save Earth Save Earth ' |
| in (Membership) | The operator displays 1 if the string contains the given character or the sequence of characters. | >>>A='Save Earth'<br>>>> 'S' in A<br>True<br>>>>'Save' in A<br>True<br>>>'SE' in A<br>False |
| not in | The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of **in** operator discussed above) | >>>'SE' not in 'Save Earth'<br>True<br>>>>'Save ' not in 'Save Earth'<br>False |
| range (*start*, *stop*[, *step*]) | **This function is already discussed in previous chapter.** | |
| Slice[n:m] | The Slice[n : m] operator extracts sub parts from the strings. | >>>A='Save Earth'<br>>>> print A[1:3]<br>av<br>The print statement prints the substring starting from subscript 1 and ending at subscript 3 but not including subscript 3 |

## More on string Slicing

Consider the given figure

| **String A** | S | A | V | E | | E | A | R | T | H |
|---|---|---|---|---|---|---|---|---|---|---|
| **Positive Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Negative Index** | -10 | -9 | -9 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Let's understand Slicing in strings with the help of few examples.

**Example**

```
>>>A='Save Earth'
>>> print A[1:3]
av
```

The print statement prints the substring starting from subscript 1 and ending at subscript 3 .

**Example**

```
>>>print A[3:]
'e Earth'
```

Omitting the second index, directs the python interpreter to extract the substring till the end of the string

**Example**

```
>>>print A[:3]
Sav
```

Omitting the first index, directs the python interpreter to extract the substring before the second index starting from the beginning.

**Example**

>>>print A[:]

'Save Earth'

Omitting both the indices, directs the python interpreter to extract the entire string starting from 0 till the last index

**Example**

>>>print A[-2:]

'th'

For negative indices the python interpreter counts from the right side (also shown above). So the last two letters are printed.

**Example**

>>>Print A[:-2]

'Save Ear'

Omitting the first index, directs the python interpreter to start extracting the substring form the beginning. Since the negative index indicates slicing from the end of the string. So the entire string except the last two letters is printed.

**Note:** Comparing strings using relational operators has already been discussed in the previous chapter

## String methods & built in functions

| Syntax | Description | Example |
|---|---|---|
| len() | Returns the length of the string. | >>>A='Save Earth'<br>>>> print len(A)<br>>>>10 |
| capitalize() | Returns the exact copy of the string with the first letter in | >>>str='welcome' |

| | | |
|---|---|---|
| | upper case | >>>print str.capitalize()<br>Welcome |
| **find**(*sub*[, *start*[, *end*]]) | The function is used to search the first occurrence of the substring in the given string. It returns the index at which the substring starts. It returns -1 if the substring does occur in the string. | >>>str='mammals'<br>>>>str.find('ma')<br>0<br>**On omitting the start parameters, the function starts the search from the beginning.**<br>>>>str.find('ma',2)<br>3<br>>>>str.find('ma',2,4)<br>-1<br>**Displays -1 because the substring could not be found between the index 2 and 4-1**<br>>>>str.find('ma',2,5)<br>3 |
| isalnum() | Returns True if the string contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc. | >>>str='Save Earth'<br>>>>str.isalnum()<br>False<br>The function returns False as space is an alphanumeric character.<br>>>>'Save1Earth'.isalnum()<br>True |
| isalpha() | Returns True if the string contains only letters. Otherwise return False. | >>> 'Click123'.isalpha()<br>False<br>>>> 'python'.isalpha()<br>True |
| isdigit() | Returns True if the string | >>>print str.isdigit() |

| | | |
|---|---|---|
| | contains only numbers. Otherwise it returns False. | false |
| lower() | Returns the exact copy of the string with all the letters in lowercase. | >>>print str.lower()<br>'save earth' |
| islower() | Returns True if the string is in lowercase. | >>>print str.islower()<br>True |
| isupper() | Returns True if the string is in uppercase. | >>>print str.isupper()<br>False |
| upper() | Returns the exact copy of the string with all letters in uppercase. | >>>print str.upper()<br>WELCOME |
| lstrip() | Returns the string after removing the space(s) on the left of the string. | >>> print str<br>Save Earth<br>>>>str.lstrip()<br>'Save Earth'<br>>>>str='Teach India Movement'<br>>>> print str.lstrip("T")<br>each India Movement<br>>>> print str.lstrip("Te")<br>ach India Movement<br>>>> print str.lstrip("Pt")<br>Teach India Movement<br>**If a string is passed as argument to the lstrip() function, it removes those characters from the left of the string.** |
| rstrip() | Returns the string after removing the space(s) on the | >>>str='Teach India Movement'<br>>>> print str.rstrip() |

| | | |
|---|---|---|
| | right of the string. | Teach India Movement |
| isspace() | Returns True if the string contains only white spaces and False even if it contains one character. | >>> str=' '<br>>>> print str.isspace()<br>True<br>>>> str='p'<br>>>> print str.isspace()<br>False |
| istitle() | Returns True if the string is title cased. Otherwise returns False | >>> str='The Green Revolution'<br>>>> str.istitle()<br>True<br>>>> str='The green revolution'<br>>>> str.istitle()<br>False |
| replace(old, new) | The function replaces all the occurrences of the old string with the new string | >>>str='hello'<br>>>> print str.replace('l','%')<br>He%%o<br>>>> print str.replace('l','%%')<br>he%%%%o |
| join () | Returns a string in which the string elements have been joined by a separator. | >>> str1=('jan', 'feb' ,'mar')<br>>>>str='&"<br>>>> str.join(str1)<br>'jan&feb&mar' |
| swapcase() | Returns the string with case changes | >>> str='UPPER'<br>>>> print str.swapcase()<br>upper<br>>>> str='lower'<br>>>> print str.swapcase() |

| | | |
|---|---|---|
| | | LOWER |
| partition(sep) | The function partitions the strings at the first occurrence of separator, and returns the strings partition in three parts i.e. before the separator, the separator itself, and the part after the separator. If the separator is not found, returns the string itself, followed by two empty strings | >>> str='The Green Revolution'<br>>>> str.partition('Rev')<br>('The Green ', 'Rev', 'olution')<br>>>> str.partition('pe')<br>('The Green Revolution', '', '')<br>>>> str.partition('e')<br>('Th', 'e', ' Green Revolution') |
| split([sep[, maxsplit]]) | The function splits the string into substrings using the separator. The second argument is optional and its default value is zero. If an integer value N is given for the second argument, the string is split in N+1 strings. | >>>str='The$earth$is$what$we$all $have$in$common.'<br>>>> str.split($,3)<br>SyntaxError: invalid syntax<br>>>> str.split('$',3)<br>['The', 'earth', 'is', 'what$we$all$have$in$common.']<br>>>> str.split('$')<br>['The', 'earth', 'is', 'what', 'we', 'all', 'have', 'in', 'common.']<br>>>> str.split('e')<br>['Th', ' Gr', '', 'n R', 'volution']<br>>>> str.split('e',2)<br>['Th', ' Gr', 'en Revolution'] |

**Note:** In the table given above, len( ) is a built in function and so we don't need import the string module. For all other functions ***import string*** statement is required for their successful execution.

Let's discuss some interesting strings constants defined in string module:

### string.ascii_uppercase

The command displays a string containing uppercase characters.

**Example**

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

### string.ascii_lowercase

The command displays a string containing all lowercase characters.

**Example**

```
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

### string.ascii_letters

The command displays a string containing both uppercase and lowercase characters.

```
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

### string.digits

The command displays a string containing digits.

```
>>> string.digits
'0123456789'
```

### string.hexdigits

The command displays a string containing hexadecimal characters.

```
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

### string.octdigits

The command displays a string containing octal characters.

```
>>> string.octdigits

'01234567'
```

### string.punctuations

The command displays a string containing all the punctuation characters.

```
>>> string.punctuations

'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}-'
```

### string.whitespace

The command displays a string containing all ASCII characters that are considered whitespace. This includes the characters space, tab, linefeed, return, formfeed, and vertical tab.

```
>>> string.whitespace

'\t\n\x0b\x0c\r '
```

### string.printable

The command displays a string containing all characters which are considered printable like letters, digits, punctuations and whitespaces.

```
>>> string.printable

'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!
"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}- \t\n\r\x0b\x0c'
```

**Note:** Import string module to get the desired results with the commands mentioned above.

## Programs using string functions and operators

**1. Program to check whether the string is a palindrome or not.**

```
defpalin():
    str=input("Enter the String")
    l=len(str)
    p=l-1
    index=0
    while (index<p):
        if(str[index]==str[p]):
            index=index+1
            p=p-1
    else:
        print "String is not a palidrome"
        break
    else:
    print "String is a Palidrome"
```

**2. Program to count no of 'p' in the string pineapple.**

```
def lettercount():
    word = 'pineapple'
    count = 0
    for letter in word:
        if letter == 'p':
        count = count + 1
    print(count)
```

## Regular expressions and Pattern matching

A regular expression is a sequence of letters and some special characters (also called meta characters). These special characters have symbolic meaning. The sequence formed by using meta characters and letters can be used to represent a group of patterns.

Let's start by understanding some meta characters.

**For example**

str= "Ram$"

The pattern "Ram$" is known as a regular expression. The expression has the meta character '$'. Meta character '$' is used to match the given regular expression at the end of the string. So the regular expression would match the string 'SitaRam' or 'HeyRam' but will not match the string 'Raman'.

Consider the following codes:

```
def find():
    import re
    string1='SitaRam'
    if
    re.search('Ram$',string1):
        print "String Found"
    else :
        print" No Match"
```

Output:

**String Found**

```
def find():
   import re
   string1='SitaRam'
   if re.search('Sita$',string1):
      print "String Found"
   else :
      print" No Match"
```

Output

**No Match**

As shown in the above examples, Regular expressions can be used in python for matching a particular pattern by importing the re module.

**Note:** re module includes functions for working on regular expression.

Now let's learn how the meta characters are used to form regular expressions.

| S.No | Meta character | Usage | Example |
|---|---|---|---|
| 1 | [ ] | Used to match a set of characters. | [ram]<br>The regular expression would match any of the characters r, a, or m.<br>[a-z]<br>The regular expression would match only lowercase characters. |
| 2 | ^ | Used to complementing a set of characters | [^ram]<br>The regular expression would match any other characters than<br>r, a or m. |
| 3 | $ | Used to match the end of string only | Ram$<br>The regular expression would match Ram in SitaRam but will not match Ram in Raman |
| 4 | * | Used to specify that the previous character can be matched zero or more times. | wate*r<br>The regular expression would match strings like watr, wateer, wateeer and so on. |
| 5 | + | Used to specify that the previous character can be matched one or more times. | wate+r<br>The regular expression would match strings like water, wateer, wateeer and so on. |

| | | | |
|---|---|---|---|
| 6 | ? | Used to specify that the previous character can be matched either once or zero times | wate?r<br>The regular expression would only match strings like watr or water |
| 7 | { } | The curly brackets accept two integer value s. The first value specifies the minimum no of occurrences and second value specifies the maximum of occurrences | wate{1,4}r<br>The regular expression would match only strings water, wateer, wateeer or wateeeer |

Let's learn about few functions from re module

### re.compile()

The re.compile( ) function will compile the pattern into pattern objects. After the compilation the pattern objects will be able to access methods for various operations like searching and subsitutions

**Example**

```
import re
p=re.compile('hell*o')
```

### re.match()

The match function is used to determine if the regular expression (RE) matches at the beginning of the string.

### re.group()

The group function is used to return the string matched the RE

**Example**

```
>>>P=re.compile('hell*o')
>>>m=re.match('hell*o', ' hellooooo world')
>>>m.group()
'hello'
```

### re.start()

The start function returns the starting position of the match.

### re.end()

The end function returns the end position of the match.

### re.span()

The span function returns the tuple containing the (start, end) positions of the match

**Example**

```
>>> import re
>>> P=re.compile('hell*o')
>>> m=re.match('hell*o', 'hellooooo world')
>>> m.start()
0
>>> m.end()
5
>>> m.span()
(0, 5)
```

### re.search()

The search function traverses through the string and determines the position where the RE matches the string

**Example**

```
>>> m=re.search('hell*o', 'favorite words hellooooo world')
>>> m.start()
15
>>> m.end()
```

```
20
>>> m.group()
'hello'
>>> m.span()
(15, 20)
```

### Re.findall()

The function determines all substrings where the RE matches, and returns them as a list.

**Example**

```
>>> m=re.findall('hell*o', 'hello my favorite words hellooooo world')
>>> m
['hello', 'hello']
```

### re.finditer()

The function determines all substrings where the RE matches, and returns them as an iterator.

**Example**

```
>>> m=re.finditer('hell*o', 'hello my favorite words hellooooo world')
>>> m
<callable-iterator object at 0x0000000002E4ACF8>
>>> for match in m:
print match.span()
(0, 5)
(24, 29)
```

As shown in the above example, m is a iterator. So m is used in the for loop.

**Script 1: Write a script to determine if the given substring is present in the string.**

```
def search_string():
    import re
    substring='water'
    search1=re.search(substring,'Water water everywhere but not a drop to drink')
    if search1:
        position=search1.start()
        print "matched", substring, "at position", position
    else:
        print "No match found"
```

**Script 2: Write a script to determine if the given substring (defined using meta characters) is present in the given string**

```
def metasearch():
  import re
  p=re.compile('sing+')
  search1=re.search(p,'Some singers sing well')
  if search1:
    match=search1.group()
    index=search1.start()
    lindex=search1.end()
    print "matched", match, "at index", index ,"ending at" ,lindex
  else:
    print "No match found"
```

## EXERCISE

1. Input a string "Green Revolution". Write a script to print the string in reverse.
2. Input the string "Success". Write a script of check if the string is a palindrome or not
3. Input the string "Successor". Write a script to split the string at every occurrence of the letter s.
4. Input the string "Successor". Write a script to partition the string at the occurrence of the letter s. Also Explain the difference between the function split( ) and partition().
5. Write a program to print the pyramid.

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

6. What will be the output of the following statement? Also justify for answer.

```
>>> print 'I like Gita\'s pink colour dress'.
```

7. Give the output of the following statements

```
>>> str='Honesty is the best policy'
>>> str.replace('o','*')
```

8. Give the output of the following statements

```
>>> str='Hello World'
>>>str.istiltle()
```

9. Give the output of the following statements.

```
>>> str="Group Discussion"
>>> print str.lstrip("Gro")
```

10. Write a program to print alternate characters in a string. Input a string of your own choice.

11. Input a string 'Python'. Write a program to print all the letters except the letter'y'.

12. Consider the string str="Global Warming"

    Write statements in python to implement the following

    a) To display the last four characters.

    b) To display the substring starting from index 4 and ending at index 8.

    c) To check whether string has alphanumeric characters or not.

    d) To trim the last four characters from the string.

    e) To trim the first four characters from the string.

    f) To display the starting index for the substring 'Wa'.

    g) To change the case of the given string.

    h) To check if the string is in title case.

    i) To replace all the occurrences of letter 'a' in the string with '*'

13. Study the given script

```
def metasearch():
    import re
    p=re.compile('sing+')
    search1=re.search(p,'Some singers sing well')
    if search1:
        match=search1.group()
        index=search1.start()
        lindex=search1.end()
        print "matched", match, "at index", index ,"ending at", lindex
else:
```

```
        print "No match found"
```

What will be the output of the above script if search() from the re module is replaced by match () of the re module. Justify your answer

14. What will be the output of the script mentioned below? Justify your answer.

```
def find():
        import re
        p=re.compile('sing+')
        search1=p.findall('Some singer sing well')
        print search1
```

15. Rectify the error (if any) in the given statements.

```
>>> str="Hello World"
>>> str[5]='p'
```

UNIT 4

# Chapter 2

# Lists



***After studying this lesson, students will be able to:***

- *Understand the concept of mutable sequence types in Python.*
- *Appreciate the use of list to conveniently store a large amount of data in memory.*
- *Create, access & manipulate list objects*
- *Use various functions & methods to work with list*
- *Appreciate the use of index for accessing an element from a sequence.*

## Introduction

Like a String, list also is sequence data type. It is an ordered set of values enclosed in square brackets []. Values in the list can be modified, i.e. it is mutable. As it is set of values, we can use index in square brackets [] to identify a value belonging to it. The values that make up a list are called its elements, and they can be of any type.

We can also say that list data type is a container that holds a number of elements in a given order. For accessing an element of the list, indexing is used.

**Its syntax is**:

**Variable name [index]** (variable name is name of the list).

It will provide the value at 'index+1' in the list. Index here, has to be an integer value- which can be positive or negative. Positive value of index means counting forward from beginning of the list and negative value means counting backward from end of the list. Remember the result of indexing a list is the value of type accessed from the list.

| Index value | Element of the list |
|---|---|
| 0, -size | 1st |
| 1, -size +1 | 2nd |

| | |
|---|---|
| 2, -size +2 | 3rd |
| .<br>.<br>. | |
| size -2, -2 | 2nd last |
| size -1, -1 | last |

Please note that in the above example size is the total number of elements in the list.

Let's look at some example of simple list:

i) >>>L1 = [1, 2, 3, 4] # list of 4 integer elements.

ii) >>>L2 = ["Delhi", "Chennai", "Mumbai"] #list of 3 string elements.

iii) >>>L3 = [ ] # empty list i.e. list with no element

iv) >>>L4 = ["abc", 10, 20] # list with different types of elements

v) >>>L5 = [1, 2, [6, 7, 8], 3] # A list containing another list known as nested list

You will study about Nested lists in later parts of the chapter.

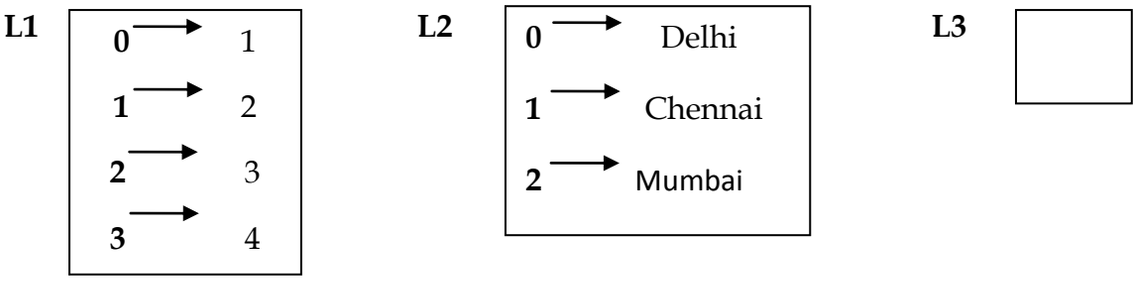
To change the value of element of list, we access the element & assign the new value.

**Example**

```
>>>print L1          # let's get the values of list before change
>>> L1 [2] = 5
>>> print L1         # modified list
[1, 2, 5, 4]
```

Here, 3rd element of the list (accessed using index value **2**) is given a new value, so instead of **3** it will be **5.**

State diagram for the list looks like:




**Note:** List index works the same way as String index, which is:

- An integer value/expression can be used as index.
- An Index Error appears, if you try and access element that does not exist in the list.
- An index can have a negative value, in that case counting happens from the end of the list.

## Creating a list

List can be created in many ways:

i) By enclosing elements in [ ], as we have done in above examples.

ii) Using other Lists

**Example**

L5=L1 [:]

Here L5 is created as a copy of L1.

>>>print L5

L6 = L1 [0:2]

>>>print L6

will create L6 having first two elements of L1.

iii) List comprehension

**Example**

>>>n = 5

```
>>>l = range(n)
>>>print l
[0, 1, 2, 3, 4]
```

**Example**

```
>>> S= [x**2 for x in range (10)]
>>> print S
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In mathematical terms, S can be defined as S = {$x^2$ for: x in (0.....9)}. So, we can say that list comprehension is short-hand for creating list.

**Example**

```
>>> A = [3, 4, 5]
>>> B = [value *3 for value in A]
```

Here B will be created with the help of A and its each element will be thrice of element of A.

```
>>> print B
[9, 12, 15]
```

Comprehensions are functionally equivalent to wrting as:

```
>>>B = [ ]
>>>for i in A
B. append (i*3)
```

Similarly, other comprehensions can be expended.

**Example**

```
>>> print B
[9, 12, 15]
```

Let's create a list of even numbers belonging to '**S**' list:

```
>>>C = [i for i in S if i % 2 = = 0]
```

```
>>>print C
[0, 4, 16, 36, 64]
```

iv) Using built-in object

L = list ( ) will create an empty list

**Example**

```
>>>l = list ( )
>>>print l
[ ] # empty list
```

Or

L = list (sequence)

**Example**

```
>>>L = list [(1, 2, 3, 4)]
>>>print L
[1, 2, 3, 4]
```

A single new list is created every time, you execute [ ]. We have created many different lists each using [ ]. But if a list is assigned to another variable, a new list is not created.

i) A=B=[ ]

Creates one list mapped to both A & B

**Example**

```
>>>A = B = [10, 20, 30]
>>> print A, B
[10, 20, 30] [10, 20, 30]
```

ii) A = [ ]

B = A

Will also create one list mapped to both

**Example**

```
>>> A = [1, 2, 3]
>>> B = A
>>> print A, B
[1, 2, 3] [1, 2, 3]
```

## Accessing an element of list

For accessing an element, we use index and we have already seen example doing so. To access an element of list containing another list, we use pair of index. Lets access elements of L5 list. Also a sub-list of list can be accessed using list slice.

### List Slices

Slice operator works on list also. We know that a slice of a list is its sub-list. For creating a list slice, we use

[n:m] operator.

```
>>>print L5 [0]
1
>>>print L5 [2]
[6, 7, 8]
```

as the 3rd element of this list is a list. To access a value from this sub-list, we will use

```
>>>print L5 [2] [0]
6
>>>print L5 [2] [2]
8
```

This will return the part of the list from $n^{th}$ element to $m^{th}$ element, including the first element but excluding the last element. So the resultant list will have m-n elements in it.

```
>>> L1 [1:2]
```

will give

```
[2]
```

Slices are treated as boundaries, and the result will contain all the elements between boundaries.

**Its Syntax is:**

**seq = L [start: stop: step]**

Where start, stop & step- all three are optional. If you omit first index, slice starts from '0' and omitting of stop will take it to end. Default value of step is 1.

**Example**

For list L2 containing ["Delhi", "Chennai", "Mumbai"]

```
>>>L2 [0:2]
["Delhi", "Chennai"]
```

**Example**

```
>>>list = [10, 20, 30, 40, 50, 60]
>>> list [::2]    # produce a list with every alternate element
[10, 30, 50]
>>>list [4:]     # will produce a list containing all the elements from 5th position till end
[50, 60]
```

**Example**

```
>>>list [:3]
[10, 20, 30]
>>>list [:]
[10, 20, 30, 40, 50, 60]
```

**Example**

```
>>> list [-1]    # '-1' refers to last elements of list
60
```

will produce a list with every other element

**Note:** Since lists are mutable, it is often recommended to make a copy of it before performing operation that change a list.

## Traversing a List

Let us visit each element (traverse the list) of the list to display them on screen. This can be done in many ways:

(i)
```
i = 0
while i < 4:
print L1 [i],
i + = 1
```
will produce following output

1 2 5 4

(ii)
```
for i in L1:
print i,
```
will also produce the same output

(iii)
```
i=0
while i < len [L1]:
print L1 [i],
i + = 1
```
**OR**
```
i= 0
L = len (L1)
while i < L :
print L1 [i],
i + = 1
```
will also produce the same output.

*Here len( ) function is used to get the length of list L1. As length of L1 is 4, i will take value from 0 to 3.*

(iv) for i in range ( len (L1)):

print L1 [i],

*Using 2nd way for transversal will only allow us to print the list, but other ways can also be used to write or update the element of the list.*

In 4th way, range ( ) function is used to generate, indices from 0 to len -1; with each iteration i gets the index of next element and values of list are printed.

**Note:** for loop in empty list is never executed:

**Example**

```
for i in [ ]:
    print i
```

Accessing list with negative index

```
i = 1
while i < len (L1):
print L1 [-i],
i += 1
```

In this case, Python will add the length of the list to index and then return the index value and accesses the desired element. In this loop execution for a positive value of 'i' L1 [-i] will result into L1 [len (L1)-i] for i=1, L1 [4-1] will be printed. So resultant of the loop will be 4 5 2.

## Appending in the list

Appending a list is adding more element(s) at the end of the list. To add new elements at the end of the list, Python provides a method append ( ).

**Its Syntax is:**

**List. append (item)**

L1. append (70)

This will add 70 to the list at the end, so now 70 will be the 5th element of the list, as it already have 4 elements.

```
>>> print L1
```

will produce following on screen

```
[1, 2, 5, 4, 70]
```

**Example**

```
>>>L4.append (30)          # will add 30 at the end of the list
>>>print L4
['abc', 10, 20, 30]
```

Using append ( ), only one element at a time can be added. For adding more than one element, extend ( ) method can be used, this can also be used to add elements of another list to the existing one.

**Example**

```
>>>A = [100, 90, 80, 50]
>>> L1. extend (A)
>>> print L1
```

will add all the elements of list 'A' at the end of the list 'L1'.

```
[1, 2, 5, 4, 70, 100, 90, 80, 50]
>>>print A
[100, 90, 80, 50]
```

**Example**

```
>>>B=[2009, 2011, 'abc']
>>>C=['xyz', 'pqr', 'mn']
>>>B.extend (c)
>>>print B
```

[2009, 2011, ‘abc’, ‘xyz’, ‘pqr’, ‘mn’]

***Remember:*** ‘A’ remains unchanged

## Updating array elements

Updating an element of list is, accomplished by accessing the element & modifying its value in place. It is possible to modify a single element or a part of list. For first type, we use index to access single element and for second type, list slice is used. We have seen examples of updations of an element of list. Lets update a slice.

**Example**

>>> L1 [1:2] = [10, 20]

>>> print L1

will produce

[1, 10, 20, 4, 70, 100, 90, 80, 50]

**Example**

>>>A=[10, 20, 30, 40]

>>>A [1:4] = [100]

>>>print A

will produce

[10, 100]

As lists are sequences, they support many operations of strings. For example, operator **+** & * results in concatenation & repetition of lists. Use of these operators generate a new list.

**Example**

>>> a= L1+L2

will produce a 3rd list **a** containing elements from L1 & then L2. **a** will contain

[1, 10, 20, 4, 70, 100, 90, 80, 50, “Delhi”, “Chennai”, “Mumbai”]

**Example**

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

**Example**

```
>>> b = L1*2
>>> print b
[[1, 10, 20, 4, 70, 100, 90, 80, 50, 1, 10, 20, 4, 70, 100, 90, 80, 50]
```

**Example**

```
>>> ['Hi!']* 3
['Hi!', 'Hi!', 'Hi!']
```

It is important to know that **'+'** operator in lists expects the same type of sequence on both the sides otherwise you get a type error.

If you want to concatenate a list and string, either you have to convert the list to string or string to list**.**

**Example**

```
>>> str([11, 12]) + "34"    or  >>>"[11,12]" + "34"
'[11, 12] 34'
>>> [11, 12] + list ("34")    or >>>[11, 12] + ["3", "4"]
  [11, 12, '3', '4']
```

## Deleting Elements

It is possible to delete/remove element(s) from the list. There are many ways of doing so:

(i) If index is known, we can use pop ( ) or del

(ii) If the element is known, not the index, remove ( ) can be used.

(iii) To remove more than one element, del ( ) with list slice can be used.

(iv) Using assignment operator

Let us study all the above methods in details:

## Pop ( )

It removes the element from the specified index, and also return the element which was removed.

**Its syntax is:**

**List.pop ([index])**

**Example**

```
>>> L1 = [1, 2, 5, 4, 70, 10, 90, 80, 50]
>>> a= L1.pop (1)          # here the element deleted will be returned to 'a'
>>> print L1
[1, 5, 4, 70, 10, 90, 80, 50]
>>> print a
2
```

*If no index value is provided in pop ( ), then last element is deleted.*

```
>>>L1.pop ( )
50
```

**del** removes the specified element from the list, but does not return the deleted value.

```
>>> del L1 [4]
>>> print L1
[1, 5, 4, 70, 90, 80]
```

## remove ( )

In case, we know the element to be deleted not the index, of the element, then remove ( ) can be used.

```
>>> L1. remove (90)
```

will remove the value 90 from the list

```
>>> print L1
[1, 5, 4, 70, 80]
```

### del () with slicing

Consider the following example:

**Examples**

```
>>> del L1 [2:4]
>>>print L1
[1, 5, 80]
```

will remove 2nd and 3rd element from the list. As we know that slice selects all the elements up to 2nd index but not the 2nd index element. So **4th element** will remain in the list.

```
>>> L5 [1:2] = [ ]
```

Will delete the slice

```
>>>print L5
[1, [6, 7, 8], 3]
```

**Note:**

(i) All the methods, modify the list, after deletions.

(ii) If an out of range index is provided with del ( ) and pop ( ), the code will result in to run-time error.

(iii) del can be used with negative index value also.

## Other functions & methods

### insert ( )

This method allows us to insert an element, at the given position specified by its index, and the remaining elements are shifted to accommodate the new element. Insert (

() requires two arguments-**index value** and **item value**.

**Its syntax is**

**list. insert (index, item)**

Index specifies the position (starting from 0) where the element is to be inserted. Item is the element to be inserted in the list. Length of list changes after insert operation.

**Example**

```
>>> L1.insert (3,100)
>>>print L1
```

will produce

[1, 5, 80, 100]

**Note:** If the index specified is greater then len (list) the object is inserted in the last and if index is less than zero, the object is inserted at the beginning.

```
>>> print len(L1)
4
>>> L1.insert (6, 29)
>>> L1.insert (-2, 46)
>>>print L1
```

will produce

[46, 1, 5, 80, 100, 29]

### reverse ( )

This method can be used to reverse the elements of the list in place

**Its syntax is:**

**list.reverse ( )**

Method does not return anything as the reversed list is stored in the same variable.

**Example**

```
>>> L1.reverse ( )
```

```
>>> print L1
```

will produce

[29, 100, 80, 5, 1, 46]

Following will also result into reversed list.

```
>>>L1 [: : -1]
```

As this slices the whole sequence with the step of -1 i.e. in reverse order.

### sort ( )

For arranging elements in an order Python provides a method **sort ( )** and a function **sorted ( )**. sort ( ) modifies the list in place and sorted ( ) returns a new sorted list.

**Its Syntax are:**

**sort ([cmp [, key [, reverse]]])**

**sorted (list [, cmp [, key [, reverse]]])**

Parameters mentioned in [ ] are optional in both the cases. These parameters allow us to customize the function/method.

**cmp**, argument allow us to override the default way of comparing elements of list. By default, sort determines the order of elements by comparing the elements in the list against each other. To overside this, we can use a user defined function which should take two values and return -1 for **'less than'**, 0 for **'equal to'** and 1 for 'greater than'.

'**Key**' argument is preferred over **'cmp'** as it produces list faster.

**Example**

The parameter **'key'** is for specifying a function that transforms each element of list before comparison. We can use predefined functions or a user defined function here. If its user defined then, the function should take a single argument and return a key which can be used for sorting purpose.

Reverse parameter can have a boolean value which is used to specify the order of arranging the elements of list. Value **'True'** for reverse will arrange the elements of list in descending order and value **'False'** for reverse will arrange the elements in ascending order. Default value of this parameter is False.

**sorted ( )** function also behaves in similar manner except for it produce a new sorted list, so original is not changed. This function can also be used to sort any iterable collection. As sort ( ) method does not create a new list so it can be little faster.

**Example**

```
>>> L1.sort ( )
>>> print L1
```

will produce

[1, 5, 29, 46, 80, 100]

```
>>> L2.sort ( )
>>> print L2
```

will produce

['Chennai', 'Delhi', 'Mumbai']

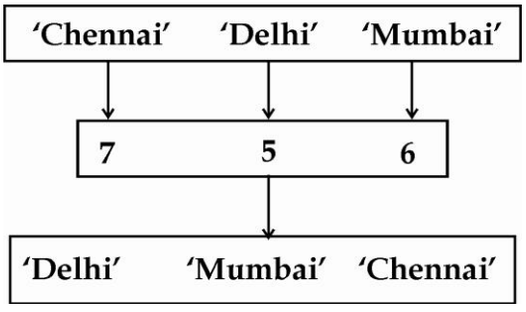
```
>>> L2.sort (key=len)
```

will produce

['Delhi', 'Mumbai', 'Chennai']

Here we have specified len ( ) built in function, as key for sorting. So the list will get sorted by the length of the strings, i.e., from shorted to longest.

sort will call len ( ) function for each element of list and then these lengths will be used for arranging elements.




```
>>> L4.sort ( )
>>> print L4
```

will produce

```
[10, 20, 30, 'abc']
>>>L4.sort (reverse = True)
['abc', 30, 20, 10]
>>> def compare (str):
...         return len (str)
>>> L2.sort (key=compare)
>>> L2
['Delhi', 'Mumbai', 'Chennai']
```

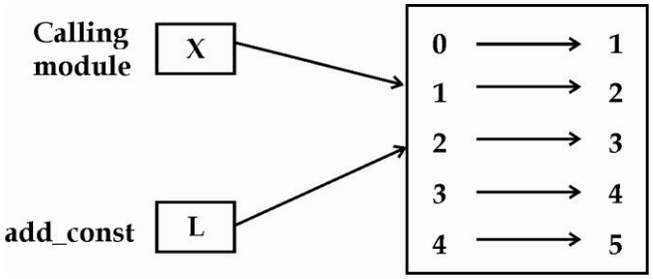
## List as arguments

When a list is passed to the function, the function gets a reference to the list. So if the function makes any changes in the list, they will be reflected back in the list.

**Example**

```
def add_Const (L):
for i in range (len (l)):
L [i] += 10
>>> X = [1, 2, 3, 4, 5]
>>> add_Const (X)
>>> print X
[11, 12, 13, 14, 15]
```

Here parameter 'L' and argument 'X' are alias for same object. Its state diagram will look like

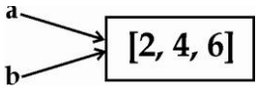



So any changes made in L will be reflected to X as lists as mutable.

**Note:** Here, it becomes important to distinguish between the operations which modifies a list and operation which creates a new list. Operations which create a new list will not affect the original (argument) list.

Let's look at some examples to see when we have different lists and when an alias is created.

```
>>> a = [2, 4, 6]
>>> b = a
```



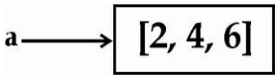

will map b to a. To check whether two variables refer to same object (i.e. having same value), we can use 'is' operator. So in our example:

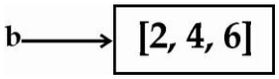
```
>>> a is b
```

will return 'True'

```
>>> a = [2, 4, 6]
>>> b = [2, 4, 6]
>>> a is b
```

False




In first example, Python created one list, reference by a & b. So there are two references to the same object b. We can say that object [2, 4, 6] is aliased as it has more than one name, and since lists are mutable. So changes made using 'a' will affect 'b'.

```
>>> a [1] = 10
>>> print b
```

will print

[2, 10, 6]

## Matrix implementation using list

We can implement matrix operation using list. Matrix operation can be implemented using nested list. List inside another list is called nested list.

**Its syntax is:**

**a=[[random.random() for row in range(number of row)]for col in range(number of column)]**

Here random function is used. So we need to import random file.

**Example**

Write a program to input any matrix with mXn, and print the number on the output screen in matrix format.

### Matrix creation

**Program 1**

```
m=input ("Enter total number of rows")
n=input ("Enter total number of columns")
l=range (m*n)
k=0
print "Input all matrix elements one after other"
for i in range(m):
for j in range(n):
l[k]=input("Enter new element")
k=k+1
print "output is"
k=0
for i in range(m):
for j in range(n):
print l[k],'\t',
k=k+1
print
```

**Output**

```
>>>
Enter total number of rows3
Enter total number of columns3
Input all matrix elements one after other
Enter new element10
Enter new element20
Enter new element30
Enter new element40
Enter new element50
Enter new element60
Enter new element70
Enter new element80
Enter new element90
output is
10      20      30
40      50      60
70      80      90
>>>
```

**Program 2**

```
import random
m=input("Enter total number of rows in the first matrix")
n=input("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)]for col in range(n)]
print "Enter all elements one after other"
for i in range(m):
```

```
for j in range(n):
a[i][j]=input()
print "output is"
for i in range(m):
for j in range(n):
print a[i][j],'\t',
print
```

**Output**

```
>>>
Enter total number of rows in the first matrix3
Enter total number of columns in the first matrix3
Enter all elements one after other
1
2
3
4
5
6
7
8
9
output is
1       2       3
4       5       6
7       8       9
>>>
```

## Matrix Addition

**Write a program to input any two matrices and print sum of matrices.**

```
import random
m1=input ("Enter total number of rows in the first matrix")
n1=input ("Enter total number of columns in the first matrix")
a=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m1):
for j in range(n1):
a[i][j]=input()
m2=input("Enter total number of rows in the second matrix")
n2=input("Enter total number of columns in the second matrix")
b=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(2):
for j in range(2):
b[i][j]=input()
c=[[random.random()for row in range(m1)]for col in range(n1)]
if ((m1==m2) and (n1==n2)):
print "output is"
for i in range(m1):
for j in range(n1):
c[i][j]=a[i][j]+b[i][j]
print c[i][j],'\t',
print
else
print “Matrix addition not possible”
```

**Output**

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first matrix2
1
1
1
1
Enter total number of rows in the second matrix2
Enter total number of columns in the second matrix2
2
2
2
2
output is
3          3
3          3
```

**Example**

**Write a program to input any two matrices and print product of matrices.**

```
import random
m1=input ("Enter total number of rows in the first matrix")
n1=input ("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m1):
for j in range(n1):
a[i][j]=input()
```

```
m2=input ("Enter total number of rows in the second matrix")
n2=input ("Enter total number of columns in the second matrix")
b=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m2):
for j in range(n2):
b[i][j]=input()
c=[[random.random()for row in range(m1)]for col in range(n2)]
if (n1==m2):
for i in range(m1):
for j in range(n2):
c[i][j]=0
for k in range(n1):
c[i][j]+=a[i][k]*b[k][j]
print c[i][j],'\t',
print
else:
print "Multiplication not possible"
```

**Output**

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first  matrix2
1
1
1
1
Enter total number of rows in the second matrix2
```

```
Enter total number of columns in the second matrix2
2
2
2
2
4          4
4          4
>>>
```

**Example**

**Write a program to input any matrix and print both diagonal values of the matrix.**

```
import random
m=input ("Enter total number of rows in the first matrix")
n=input ("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)] for col in range(n)]
if (m==n):
for i in range(m):
for j in range(n):
a[i][j]=input()
print "First diagonal"
for i in range(m):
print a[i][i],'\t',
print
k=m-1
print "Second diagonal"
for j in range(m):
print a[j][k],'\t',
```

```
    k-=1
    else:
    print "Diagonal values are not possible"
```

**Output**

```
>>>
Enter total number of rows in the first matrix3
Enter total number of columns in the first matrix3
1
2
3
4
5
6
7
8
9
First diagonal
1    5    9
Second diagonal
3     5    7
>>>
```

## Functions with list

We can pass list value to function. Whatever modification we are doing with in function will affect list.

**Example**

**Write a program to pass any list and to arrange all numbers in descending order.**

```
def arrange (l,n):
for i in range(n-1):
for j in range(n-i-1):
if l[j]>l[j+1]:
temp=l[j]
l[j]=l[j+1]
l[j+1]=temp
```

**Output**

```
>>>
>>> l=[7,5,8,2,9,10,3]
>>> arrange (l)
>>> print l
[10, 9, 8, 7, 5, 3, 2]
>>>
```

Function pass nested list also:

**Example**

**Write a program to input nXm matrix and find sum of all numbers using function.**

**Function:**

```
def summat(a,m,n):
s=0
for i in range(m):
for j in range(n):
s+=a[i][j]
return s
```

**Note:** This function is stored in mataddition.py

## Function call

```
import random
import mataddition
m=input("Enter total number of rows in the first matrix")
n=input("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)]for col in range(n)]
for i in range(m):
for j in range(n):
a[i][j]=input()
s=mataddition.summat(a,m,n)
print s
```

**Output**

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first matrix2
1
2
3
4
10
>>>
```

**Example**

```
# Accessing the elements of a sublist
a = [[1, 2, 3], [4, 5], [6, 7, 8]]
count = -1
for list in a:
```

```
count + = 1
print “elements of the list at index”, count, “are:”
for item in list:
print item,
print
```

will produce the result

```
elements of the list at index 0 are
1 2 3
elements of the list at index 1 are
4 5
elements of the list at index 2 are
6 7 8
```

## EXERCISE

1. Define list

2. What is the output of the following code:

   a) print type ([1,2])

      (i) <type 'complex'>

      (ii) <type 'int'>

      (iii) <type 'list'>

   b) a= [1, 2, 3, None, ( ), [ ]}

      print len(a)

      (i) Syntax error (ii) 4

      (iii) 5 (iv) 6

      (v) 7

3. Write the output from the following code:

```
A=[2,4,6,8,10]
L=len(L)
S=0
for I in range(1,L,2):
S+=A[I]
print "Sum=",S
```

4. Find the errors from the following program

```
n=input (Enter total number of elements)
l=range(n)
print l
for i in (n);
l[i]=input("enter element")
```

```
print "All elements in the list on the output screen"
for i on range(n):
print l[i]
```

5. Write a function group of list (list, size) that takes a list and splits into smaller list of given size.

6. Write a function to find all duplicates in the list.

7. For each of the expression below, specify its type and value. If it generates error, write error.

   Assume that expressions are evaluated in order.

   x= [1, 2, [3, 'abc', 4], 'Hi']

   (i) x[0]

   (ii) x[2]

   (iii) x[-1]

   (iv) x[0:1]

   (v) 2 in x

   (vi) x[0]=8

8. For each of the expression below, specify its type and value. If it generates error, write error:

   List A= [1, 4, 3, 0]

   List B= ['x', 'z', 't', 'q']

   (i) List A.sort ( )

   (ii) List A

   (iii) List A.insert (0, 100)

   (iv) List A.remove (3)

   (v) List A.append (7)

   (vi) List A+List B

(vii) List B.pop ( )

(viii) List A.extend ([4, 1, 6, 3])

## LAB EXERCISE

1. We can use list to represent polynomial.

   For Example

   $p(x) = -13.39 + 17.5\,x + 3\,x^2 + x^4$

   can be stored as

   [-13.39, 17.5, 3, 1.0]

   Here 'index' is used to represent power of 'x' and value at the index used to represent the coefficient of the term.

   Write a function to evaluate the polynomial for a given 'x'.

2. Write a function that takes a list of numbers and returns the cumulative sum; that is, a new list where the its element is the sum of the first i+1 elements from the original list. For example, the cumulative sum of [1, 2, 3] is [1, 3, 6].

3. Write a function called *chop* that takes a list and modifies it, removing the first and last elements, and returns *None*. Then write a function called *middle* that takes a list and returns a new list that contains all but the first and last elements.

4. Write a function called *is_sorted* that takes a list as a parameter and returns *True* if the list is sorted in ascending order and *False* otherwise. You can assume (as a precondition) that the elements of the list can be compared with the relational operators <, >, etc.

   For example, *is_sorted ([1, 2, 2])* should return *True* and *is_sorted (['b', 'a'])* should return *False.*

5. Write a function called *remove_duplicates* that takes a list and returns a new list with only the unique elements from the original. Hint: they don't have to be in the same order.

6. Write a function that takes in two sorted lists and merges them. The lists may not be of same length and one or both may be empty. Don't use any Python built-in methods or functions.

7. Create a list that contains the names of 5 students of your class. (Do not ask for input to do so)

   (i) Print the list

   (ii) Ask the user to input one name and append it to the list

   (iii) Print the list

   (iv) Ask user to input a number. Print the name that has the number as index (Generate error message if the number provided is more than last index value).

   (v) Add "Kamal" and "Sanjana" at the beginning of the list by using '+'.

   (vi) Print the list

   (vii) Ask the user to type a name. Check whether that name is in the list. If exist, delete the name, otherwise append it at the end of the list.

   (viii) Create a copy of the list in reverse order

   (ix) Print the original list and the reversed list.

   (x) Remove the last element of the list.

8. Use the list of student names from the previous exercise. Create a for loop that asks the user for every name whether they would like to keep the name or delete it. Delete the names which the user no longer wants. Hint: you cannot go through a list using a for loop and delete elements from the same list simultaneously because in that way the for loop will not reach all elements. You can either use a second copy of the list for the loop condition or you can use a second empty list to which you append the elements that the user does not want to delete.

9. Write a function to find product of the element of a list. What happens when the function is called with list of strings?

10. Write a program to input NXM matrix and find sum of all even numbers in the matrix.

11. Write a program to print upper triangle matrix.

12. Write a program to print lower triangle matrix.

13. Write a program to find sum of rows and columns of the matrix.

# Chapter 3

# Dictionaries



***After studying this lesson, the students will be able to***

- *understand the need of dictionaries;*
- *solve problems by using dictionaries;*
- *get clear idea about dictionaries functions; and*
- *understand the difference between list and dictionary.*

## What is dictionary?

A dictionary is like a list, but more in general. In a list, index value is an integer, while in a dictionary index value can be any other data type and are called keys. The key will be used as a string as it is easy to recall. A dictionary is an extremely useful data storage construct for storing and retrieving all key value pairs, where each element is accessed (or indexed) by a unique key. However, dictionary keys are not in sequences and hence maintain no left-to right order.

## Key-value pair

We can refer to a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps a value. The association of a key and a value is called a **key-value pair**.

**Syntax:**

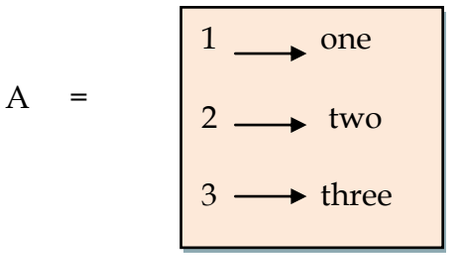
**my_dict = {'key1': 'value1','key2': 'value2','key3': 'value3'…'keyn': 'valuen'}**

**Note:** Dictionary is created by using curly brackets(ie. {}).

**Example**

```
>>> A={1:"one",2:"two",3:"three"}
>>> print A
{1: 'one', 2: 'two', 3: 'three'}
```

In the above example, we have created a list that maps from numbers to English words, so the keys values are in numbers and values are in strings.




Map between keys and values

**Example**

```
>>>computer={'input':'keybord','output':'mouse','language':'python','os':'windows-8',}
>>> print computer
{'input': 'keyboard', 'os': 'windows-8', 'language': 'python', 'output': 'mouse'}
>>>
```

In the above example, we have created a list that maps from computer related things with example, so here the keys and values are in strings. The order of the key-value pairs is not in same order (ie. input and output orders are not same). We can get different order of items in different computers. Thus, the order of items in a dictionary is unpredictable.

**Example**

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursday','fri':'Friday','sat':'Saturday'}
>>> print D
{'wed': 'Wednesday', 'sun': 'Sunday', 'thu': 'Thursday', 'tue': 'Tuesday', 'mon': 'Monday', 'fri': 'Friday', 'sat': 'Saturday'}
```

## Creation, initializing and accessing the elements in a Dictionary

The function dict ( ) is used to create a new dictionary with no items. This function is called built-in function. We can also create dictionary using {}.

```
>>> D=dict()
>>> print D
{}
```

{} represents empty string. To add an item to the dictionary (empty string), we can use square brackets for accessing and initializing dictionary values.

**Example**

```
>>> H=dict()
>>> H["one"]="keyboard"
>>> H["two"]="Mouse"
>>> H["three"]="printer"
>>> H["Four"]="scanner"
>>> print H
{'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
>>>
```

## Traversing a dictionary

Let us visit each element of the dictionary to display its values on screen. This can be done by using 'for-loop'.

**Example**

**Code**

```
H={'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
for i in H:
print i,":", H[i],"   ",
```

**Output**

```
>>>
Four: scanner     one: keyboard     three: printer     two: Mouse
>>>
```

OR

**Code**

```
H = {'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
print "i value","\t","H[i] value"
for i in H:
print i,"\t", H[i]
```

**Output**

```
i value   H[i] value
Four      scanner
one       keyboard
three     printer
two       Mouse
```

As said previously, the order of items in a dictionary is unpredictable.

## Creating, initializing values during run time (Dynamic allocation)

We can create a dictionary during run time also by using dict () function. This way of creation is called dynamic allocation. Because, during the run time, memory keys and values are added to the dictionary.

**Example**

Write a program to input total number of sections and class teachers' name in 11th class and display all information on the output screen.

**Code**

```
classxi=dict()
n=input("Enter total number of section in xi class")
i=1
while i<=n:
a=raw_input("enter section")
```

```
b=raw_input ("enter class teacher name")
classxi[a]=b
i=i+1
print "Class","\t","Section","\t","teacher name"
for i in classxi:
print "XI","\t",i,"\t",classxi[i]
```

**Output**

```
>>>
Enter total number of section in xi class3
enter sectionA
enter class teacher nameLeena
enter sectionB
enter class teacher nameMadhu
enter sectionC
enter class teacher nameSurpreeth
Class    Section    teacher name
XI       A          Leena
XI       C          Surpreeth
XI       B          Madhu
>>>
```

## Appending values to the dictionary

We can add new elements to the existing dictionary, extend it with single pair of values or join two dictionaries into one. If we want to add only one element to the dictionary, then we should use the following method.

**Syntax:**

**Dictionary name [key]=value**

**Example**

```
>>> a={"mon":"monday","tue":"tuesday","wed":"wednesday"}
>>> a["thu"]="thursday"
>>> print a
{'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday', 'tue': 'tuesday'}
>>>
```

## Merging dictionaries: An update ( )

Two dictionaries can be merged in to one by using update ( ) method. It merges the keys and values of one dictionary into another and overwrites values of the same key.

**Syntax:**

**Dic_name1.update (dic_name2)**

Using this dic_name2 is added with Dic_name1.

**Example**

```
>>> d1={1:10,2:20,3:30}
>>> d2={4:40,5:50}
>>> d1.update(d2)
>>> print d1
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
```

**Example**

```
{1: 10, 2: 30, 3: 30, 5: 40, 6: 60}  # k>>> d1={1:10,2:20,3:30}   # key 2 value is 20
>>> d2={2:30,5:40,6:60}  #key 2 value is 30
>>> d1.update(d2)
>>> print d1
```

ey 2 value is replaced with 30 in d1

## Removing an item from dictionary

We can remove item from the existing dictionary by using del key word.

**Syntax:**

**del dicname[key]**

**Example**

```
>>> A={"mon":"monday","tue":"tuesday","wed":"wednesday","thu":"thursday"}
>>> del A["tue"]
>>> print A
{'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday'}
>>>
```

## Dictionary functions and methods

### cmp ( )

This is used to check whether the given dictionaries are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first dictionary having more number of items, then it will return 1, otherwise return -1.

**Syntax:**

**cmp(d1,d2) #d1and d2 are dictionary.**

**returns 0 or 1 or -1**

**Example**

```
>>>
D1={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursd
ay','fri':'Friday','sat':'Saturday'}
>>>
D2={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursd
ay','fri':'Friday','sat':'Saturday'}
>>> D3={'mon':'Monday','tue':'Tuesday','wed':'Wednesday'}
>>> cmp(D1,D3)                    #both are not equal
```

```
1
>>> cmp(D1,D2)        #both are  equal
0
>>> cmp(D3,D1)
-1
```

### len( )

This method returns number of key-value pairs in the given dictionary.

**Syntax:**

**len(d)  #d dictionary**

returns number of items in the list.

**Example**

```
>>> H={'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
>>> len(H)
4
```

### clear ( )

It removes all items from the particular dictionary.

**Syntax:**

**d.clear( )  #d dictionary**

**Example**

```
>>> D={'mon':'Monday','tue':'Tuesday','wed':'Wednesday'}
>>> print D
{'wed': 'Wednesday', 'mon': 'Monday', 'tue': 'Tuesday'}
>>> D.clear( )
>>> print D
{}
```

### get(k, x )

There are two arguments (k, x) passed in 'get( )' method. The first argument is key value, while the second argument is corresponding value. If a dictionary has a given key (k), which is equal to given value (x), it returns the corresponding value (x) of given key (k). However, if the dictionary has no key-value pair for given key (k), this method returns the default values same as given key value. The second argument is optional. If omitted and the dictionary has no key equal to the given key value, then it returns None.

**Syntax:**

**D.get (k, x)  #D dictionary, k key and x value**

**Example**

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.get('wed',"wednesday")        # corresponding value wed
'Wednesday'
>>> D.get("fri","monday")           # default value of fri
'Friday'
>>> D.get("mon")                    # default value of mon
'Monday'
>>> D.get("ttu")                    # None
>>>
```

### has_key( )

This function returns 'True', if dictionary has a key, otherwise it returns 'False'.

**Syntax:**

**D.has_key(k)  #D dictionary and k key**

**Example**

```
>>>
```

D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda y','fri':'Friday','sat':'Saturday'}

>>> D.has_key("fri")

True

>>> D.has_key("aaa")

False

>>>

### items( )

It returns the content of dictionary as a list of key and value. The key and value pair will be in the form of a tuple, which is not in any particular order.

**Syntax:**

**D.items() # D dictionary**

**Example**

>>>

D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda y','fri':'Friday','sat':'Saturday'}

>>> D.items()

[('wed', 'Wednesday'), ('sun', 'Sunday'), ('thu', 'Thursday'), ('tue', 'Tuesday'), ('mon', 'Monday'), ('fri', 'Friday'), ('sat', 'Saturday')]

**Note:** items () is different from print command because, in print command dictionary values are written in {}

### keys()

It returns a list of the key values in a dictionary, , which is not in any particular order.

**Syntax:**

**D.keys( ) #D dictionary**

**Example**

>>>

```
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.keys()
['wed', 'sun', 'thu', 'tue', 'mon', 'fri', 'sat']
>>>
```

### values()

It returns a list of values from key-value pairs in a dictionary, which is not in any particular order. However, if we call both the items () and values() method without changing the dictionary's contents between these two (items() and values()), Python guarantees that the order of the two results will be the same.

**Syntax:**

```
D.values()      #D values
```

**Example**

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.values()
['Wednesday', 'Sunday', 'Thursday', 'Tuesday', 'Monday', 'Friday', 'Saturday']
>>> D.items()
[('wed', 'Wednesday'), ('sun', 'Sunday'), ('thu', 'Thursday'), ('tue', 'Tuesday'), ('mon',
'Monday'), ('fri', 'Friday'), ('sat', 'Saturday')]
```

## Solved Examples

1. Write a python program to input 'n' names and phone numbers to store it in a dictionary and to input any name and to print the phone number of that particular name.

**Code**

```
phonebook=dict()
```

```
n=input("Enter total number of friends")
i=1
while i<=n:
a=raw_input("enter name")
b=raw_input("enter phone number")
phonebook[a]=b
i=i+1
name=raw_input("enter name")
f=0
l=phonebook.keys()
for i in l:
if (cmp(i,name)==0):
print "Phone number= ",phonebook[i]
f=1
if (f==0):
print "Given name not exist"
```

**Output**

```
>>>
Enter total number of friends3
enter nameMona
enter phone number23456745
enter nameSonu
enter phone number45678956
enter nameRohan
enter phone number25678934
enter nameSonu
```

Phone number= 45678956

>>>

2. Write a program to input 'n' employee number and name and to display all employee's information in ascending order based upon their number.

**Code**

```
empinfo=dict()
n=input("Enter total number of employees")
i=1
while i<=n:
a=raw_input("enter number")
b=raw_input("enter name")
empinfo[a]=b
i=i+1
l=empinfo.keys()
l.sort()
print "Employee Information"
print "Employee Number",'\t',"Employee Name"
for i in l:
print i,'\t',empinfo[i]
```

**Output**

```
>>>
Enter total number of employees5
enter number555
enter nameArpit
enter number333
```

```
enter nameShilpa
enter number777
enter nameKush
enter number222
enter nameAnkita
enter number666
enter nameArun
Employee Information
Employee Number      Employee Name
222                   Ankita
333                   Shilpa
555                   Arpit
666                    Arun
777                    Kush
>>>
```

3. Write the output for the following Python codes.

```
A={1:100,2:200,3:300,4:400,5:500}
print A.items()
print A.keys()
print A.values()
```

**Output**

```
[(1, 100), (2, 200), (3, 300), (4, 400), (5, 500)]
[1, 2, 3, 4, 5]
[100, 200, 300, 400, 500]
```

4. Write a program to create a phone book and delete particular phone number using name.

**Code**

```
phonebook=dict()
n=input("Enter total number of friends")
i=1
while i<=n:
a=raw_input("enter name")
b=raw_input("enter phone number")
phonebook[a]=b
i=i+1
name=raw_input("enter name")
del phonebook[name]
l=phonebook.keys()
print "Phonebook Information"
print "Name",'\t',"Phone number"
for i in l:
print i,'\t',phonebook[i]
```

**Output**

```
>>>
Enter total number of friends5
enter nameLeena
enter phone number 9868734523
enter nameMadhu
enter phone number 9934567890
enter nameSurpreeth
```

```
enter phone number 9678543245
enter nameDeepak
enter phone number 9877886644
enter nameAnuj
enter phone number 9655442345
enter nameDeepak
Phonebook Information
Name            Phone number
Leena           9868734523
Surpreeth       9678543245
Madhu           9934567890
Anuj            9655442345
>>>
```

## EXERCISE

1. Write the code to input any 5 years and the population of any city and print it on the screen.

2. Write a code to input 'n' number of subject and head of the department and also display all information on the output screen.

3. Write the output for the following codes.

```
A={10:1000,20:2000,30:3000,40:4000,50:5000}
print A.items()
print A.keys()
print A.values()
```

4. Write a code to create customer's list with their number & name and delete any particular customer using his /her number.

5. Write a Python program to input 'n' names and phone numbers to store it in a dictionary and print the phone number of a particular name.

6. Find errors from the following codes:

```
c=dict()
n=input(Enter total number )
i=1
while i<=n
a=raw_input("enter place")
b=raw_input("enter number")
c(a)=b
i=i+1
print "place","\t","number"
for i in c:
print   i,"\t",cla[i]
```

# Chapter 4

# Tuples



***After studying this lesson, the students will be able to***

- *understand the need of Tuples;*
- *solve problems by using Tuples;*
- *get clear idea about Tuple functions; and*
- *understand the difference between list, dictionary and tuples.*

## What is a Tuple?

A tuple is a sequence of values, which can be of any type and they are indexed by integer. Tuples are just like list, but we can't change values of tuples in place. Thus tuples are immutable. The index value of tuple starts from 0.

A tuple consists of a number of values separated by commas. For example:

```
>>> T=10, 20, 30, 40
>>> print T
(10, 20, 30, 40)
```

But in the result, same tuple is printed using parentheses. To create a tuple with single element, we have to use final comma. A value with in the parenthesis is not tuple.

**Example**

```
>>> T=(10)
>>> type(T)
<type 'int'>
```

**Example**

```
>>> t=10,
>>> print t
(10,)
```

**Example**

```
>>> T=(10,20)
>>> type(T)
<type 'tuple'>
```

**Example**

Tuple with string values

```
>>> T=('sun','mon','tue')
>>> print T
('sun', 'mon', 'tue')
```

**Example**

Tuples with single character

```
>>> T=('P','Y','T','H','O','N')
>>> print T
('P', 'Y', 'T', 'H', 'O', 'N')
```

## Tuple Creation

If we need to create a tuple with a single element, we need to include a final comma.

**Example**

```
>>> t=10,
>>> print t
(10,)
```

Another way of creating tuple is built-in function tuple ().

**Syntax:**

**T = tuple()**

**Example**

```
>>> T=tuple()
```

```
>>> print T
()
```

## Add new element to Tuple

We can add new element to tuple using + operator.

**Example**

```
>>> t=(10,20,30,40)
>>> t+(60,)     # this will not create modification of t.
(10, 20, 30, 40, 60)
>>> print t
(10, 20, 30, 40)
>>> t=t+(60,) # this will do modification of t.
>>> print t
(10, 20, 30, 40, 60)
```

**Example**

Write a program to input 'n' numbers and store it in tuple.

**Code**

```
t=tuple()
n=input("Enter any number")
print " enter all numbers one after other"
for i in range(n):
a=input("enter number")
t=t+(a,)
print "output is"
print t
```

**Output**

```
>>>
Enter any number3
enter all numbers one after other
enter number10
enter number20
enter number30
output is
(10, 20, 30)
>>>
```

Another version of the above program:

**Code**

```
t=tuple()
n=input("Enter any number")
print " enter all numbers one after other"
for i in range(n):
a=input("enter number")
t=t+(a,)
print "output is"
for i in range(n):
print t[i]
```

**Output**

```
>>>
Enter any number3
enter all numbers one after other
enter number10
```

```
enter number20
enter number30
output is
10
20
30
>>>
```

We can also add new element to tuple by using list. For that we have to convert the tuple into a list first and then use append() function to add new elements to the list. After completing the addition, convert the list into tuple. Following example illustrates how to add new elements to tuple using a list.

```
>>> T=tuple()  #create empty tuple
>>> print T
()
>>> l=list(T)          #convert tuple      into list
>>> l.append(10)       #Add new elements to list
>>> l.append(20)
>>> T=tuple(l)         #convert list into tuple
>>> print T
(10, 20)
```

Initializing tuple values:

```
>>> T=(0,)*10
>>> print T
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

## Tuple Assignment

If we want to interchange (swap) any two variable values, we have to use temporary variable. For example;

```
>>> A=10
>>> B=20
>>> print A,B
10 20
>>> T=A
>>> A=B
>>> B=T
>>> print A,B
20 10
```

But in python, **tuple assignment** is more elegant:

**Example**

```
>>> T1=(10,20,30)
>>> T2=(100,200,300,400)
>>> print T1
(10, 20, 30)
>>> print T2
(100, 200, 300, 400)
>>> T1,T2=T2,T1      # swap T1 and T2
>>> print T1
(100, 200, 300, 400)
>>> print T2
(10, 20, 30)
```

The left side is a tuple of variables, while the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments.

The number of variables on the left and the number of values on the right have to be the same:

**Example**

```
>>> T1=(10,20,30)
>>> T2=(100,200,300)
>>> t3=(1000,2000,3000)
>>> T1,T2=T2,T1,t3
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    T1,T2=T2,T1,t3
ValueError: too many values to unpack
```

Here, two tuples are in the left side and three tuples are in right side. That is why, we get errors. Thus, it is required to have same number of tuples in both sides to get the correct result.

**Example**

```
>>> T1,T2,t3=t3,T1,T2
>>> print T1
(1000, 2000, 3000)
>>> print T2
(10, 20, 30)
>>> print t3
(100, 200, 300)
```

## Tuple Slices

Slice operator works on Tuple also. This is used to display more than one selected value on the output screen. Slices are treated as boundaries and the result will contain all the elements between boundaries.

**Syntax is:**

**Seq = T [start: stop: step]**

Where start, stop & step all three are optional. If we omit first index, slice starts from '0'. On omitting stop, slice will take it to end. Default value of step is 1.

**Example**

```
>>> T=(10,20,30,40,50)
>>> T1=T[2:4]
>>> print T1
(30, 40)
```

In the above example, starting position is 2 and ending position is 3(4-1), so the selected elements are 30 & 40.

```
>>> T[:]
(10, 20, 30, 40, 50)
```

Will produce a copy of the whole tuple.

```
>>> T[::2]
(10, 30, 50)
```

Will produce a Tuple with every alternate element.

```
>>> T[:3]
(10, 20, 30)
```

Will produce 0 to 2(3-1)

```
>>> T[2:]
(30, 40, 50)
```

Will produce from 2 to end.

## Tuple Functions

### cmp( )

This is used to check whether the given tuples are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first tuple is big, then it will return 1, otherwise return -1.

**Syntax:**

**cmp(t1,t2)** **#t1and t2 are tuples.**

**returns 0 or 1 or -1**

**Example**

```
>>> T1=(10,20,30)
>>> T2=(100,200,300)
>>> T3=(10,20,30)
>>> cmp(T1,T2)
-1
>>> cmp(T1,T3)
0
>>> cmp(T2,T1)
1
```

### len( )

It returns the number of items in a tuple.

**Syntax:**

**len(t)** **#t tuples**

returns number of items in the tuple.

**Example**

```
>>> T2=(100,200,300,400,500)
>>> len(T2)
5
```

### max( )

It returns its largest item in the tuple.

**Syntax:**

**max(t)  #t tuples**

returns maximum value among the given tuple.

**Example**

```
>>> T=(100,200,300,400,500)
>>> max(T)
500
```

### min( )

It returns its smallest item in the tuple.

**Syntax:**

**min(t)  #t tuples**

returns minimum value among the given tuple.

**Example**

```
>>> T=(100,200,300,400,500)
>>> min(T)
100
```

### tuple( )

It is used to create empty tuple.

**Syntax:**

**T=tuple()  #t tuples**

Create empty tuple.

**Example**

```
>>> t=tuple()
>>> print t
()
```

## Solved Examples

1. Write a program to input 5 subject names and put it in tuple and display that tuple information on the output screen.

**Code**

```
t=tuple()
print " enter all subjects one after other";
for i in range(5):
a=raw_input("enter subject")
t=t+(a,)
print "output is"
print t
```

**Output**

```
>>>
enter all subjects one after other
enter subjectEnglish
enter subjectHindi
enter subjectMaths
enter subjectScience
enter subjectSocial Science
output is
('English', 'Hindi', 'Maths', 'Science', 'Social Science')
>>>
```

2. Write a program to input any two tuples and interchange the tuple values.

**Code**

```
t1=tuple()
n=input("Total number of values in first tuple")
```

```
for i in range(n):
a=input("enter elements")
t1=t1+(a,)
t2=tuple()
m=input("Total number of values in first tuple")
for i in range(m):
a=input("enter elements")
t2=t2+(a,)
print "First Tuple"
print t1
print "Second Tuple"
print t2
t1,t2=t2,t1
print "AFTER SWAPPING"
print "First Tuple"
print t1
print "Second Tuple"
print t2
```

**Output**

```
>>>
Total number of values in first tuple3
enter elements100
enter elements200
enter elements300
Total number of values in first tuple4
enter elements10
```

enter elements20

enter elements30

enter elements40

First Tuple

(100, 200, 300)

Second Tuple

(10, 20, 30, 40)

**AFTER SWAPPING**

First Tuple

(10, 20, 30, 40)

Second Tuple

(100, 200, 300)

>>>

3. Write a program to input 'n' numbers and store it in a tuple and find maximum & minimum values in the tuple.

**Code**

```
t=tuple()
n=input("Total number of values in  tuple")
for i in range(n):
a=input("enter elements")
t=t+(a,)
print "maximum value=",max(t)
print "minimum value=",min(t)
```

**Output**

```
>>>
Total number of values in  tuple3
```

enter elements40

enter elements50

enter elements10

maximum value= 50

minimum value= 10

>>>

4. Find the output from the following code:

```
T=(10,30,2,50,5,6,100,65)
print max(T)
print min(T)
```

**Output**

100

2

5. Find the output from the following code:

```
t=tuple()
t = t +(PYTHON,)
print t
print len(t)
t1=(10,20,30)
print len(t1)
```

**Output**

('PYTHON',)

1

3

## EXERCISE

1. Write the output from the following codes;

   (i) t=(10,20,30,40,50)

   print len(t)

   (ii) t=('a','b','c','A','B')

   max(t)

   min(t)

   (iii) T1=(10,20,30,40,50)

   T2 =(10,20,30,40,50)

   T3 =(100,200,300)

   cmp(T1,T2)

   cmp(T2,T3)

   cmp(T3,T1)

   (iv) t=tuple()

   Len(t)

   (v) T1=(10,20,30,40,50)

   T2=(100,200,300)

   T3=T1+T2

   print T3

2. Write a program to input two set values and store it in tuples and also do the comparison.

3. Write a program to input 'n' employees' salary and find minimum & maximum salary among 'n' employees.

4. Find the errors from the following code:

   t=tuple{}

```
n=input(Total number of values in  tuple)
for i in range(n)
a=input("enter elements")
t=t+(a)
print "maximum value=",max(t)
print "minimum value=",min(t)
```

5. Write a program to input 'n' customers' name and store it in tuple and display all customers' names on the output screen.

6. Write a program to input 'n' numbers and separate the tuple in the following manner.

**Example**

T=(10,20,30,40,50,60)

T1 =(10,30,50)

T2=(20,40,60)

CBSE
CONTINUOUS AND COMPREHENSIVE EVALUATION
KNOW - AS YOU GROW


CBSE
CONTINUOUS AND COMPREHENSIVE EVALUATION
KNOW - AS YOU GROW