1. PYTHON: Introduction

Python is a programming language. It's used for many different applications. It's used in some high schools and colleges as an introductory programming language because Python is easy to learn, but it's also used by professional software developers at places such as Google, NASA, and Lucasfilm Ltd.

By the way, the language is named after the BBC show "Monty Python's Flying Circus" and has nothing to do with reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

Features:

It is compact and very easy to use OOP language.
It is more capable to express the purpose of the code.
It is interpreted line by line.
No need to download additional libraries.
It can run on variety of platform. Thus it is a portable language.
It is free and open source.
Variety of applications
Python allows you to split your program into modules that can be reused in other Python programs.
Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:

- the high-level data types allow you to express complex operations in a single statement;
- statement grouping is done by indentation instead of beginning and ending brackets;
- no variable or argument declarations are necessary.

Disadvantages:

- Python is not the fastest language.
- Its library is not still competent with other language like C, Perl and JAVA.
- Python interpreter is not very strong on catching 'Type-Mismatch'.
- It is not easy convertible to other programming language.

Typing Python in a text Editor:
Install Sublime Text 3 editor in the machine where you work

PYTHON fundamentals

Character set:

Letters: alphabets like A-Z and a-z
Numbers: 0-9
Special symbols: + - , * / ** \ [] {} () / / = != == < , > . ' "" ; : % !
Whitespaces: Blank spaces, tabs, carriage return, newline, formfeed etc.
Other characters: All ASCII and UNICODE characters.

Tokens:

The smallest individual unit of program is known as lexical unit or token.
Python has following tokens:

A. Keywords
B. Identifiers
C. Literals
D. Operators
E. Punctuators

A. Keywords: Keywords are special meaning and it is reserved by the programming language. Python has the following keywords.

false	Assert	del	for	in	or	while
none	Break	elif	from	is	pass	with
true	class	eElse	global	lamba	raise	yield
and	continue	except	if	nonlocal	return	

B. Identifiers: It is the name given by the user for declaring variables, constants, objects, classes, functions etc.

Naming rules of Identifiers are:

1. An identifier is an arbitrarily long sequence of letters and digits.
2. The first character must be a letter; the underscore count as a letter.
3. Python is a case sensitive.
4. Digits can be used in between.
5. It must not be a keyword.
6. No special character except underscore is allowed.

C. Literals/ Values

Literals are data items that have a fixed value. The types of literal are:

(i) String
(ii) Numeric
(iii) Boolean
(iv) Special Literal None

(i) ***String***: String literals are formed using **single quote** or **double quote** in Python. Python allows you to have certain non-graphic characters in string values. Non-graphic characters are those characters that cannot be typed directly from keyboard e.g. backspace, tab, spacebar, carriage return etc. These non graphic characters are represented by using escape sequences by using back slash (\) followed by one or more characters. Some escape sequences are; \a, \b, \\, \', \", \n, \f etc.

Types of String in Python:
(a) Single line String
(b) Multiple line String

(a) **Single Line String:** These strings are terminated in one line and enclosed within single quote or double quote. Example:
Str1 = 'Good
Morning'
The above example will show an error. To rectify it add a slash \ at the end of the first line
Str1 = 'Good \
Morning'

(b) **Multiple line string**: Multiple string can be created using two ways:

a. By adding a backslash
Str1 = 'Good \
Morning'

b. By typing the text in triple quotation mark
Example:
Str2= """ Hello
All of you
Good Morning."""

Size of String:
'\\' Size is 1
'xyz' Size is 3
"\ab" Size is 2
"Namrata\'s pen" Size is 13

(ii) **Numeric Literals**: The numeric literals in Python are:
a. int: it represents positive or negative whole numbers.
b. float: It represents real or the numbers with decimal point.
c. Complex: These numbers are in the form of a + bj where j is √-1 (Which is an imaginary numbers.)

(iii) **Boolean Literal:** The Boolean literal is used to represent one of the two Boolean values i.e. True or False.

(iv) **Special Literal None**: The special literal None is used to represent absence of the value.

D. Operators: Operators are symbols used for calculations. The following operators are used in Python:

a. Unary Operators
   i. + Unary Plus
   ii. -Unary Minus
   iii. ~ Bitwise complements
   iv. Not Logical operator
b. Binary Operators
   i. + Addition
   ii. - Subtraction
   iii. * Multiplication
   iv. / Division
   v. % Remainder
   vi. ** Exponent (raise to power)
   vii. // Floor division
c. Bitwise Operator
   i. & Bitwise AND
   ii. ^ Bitwise Exclusive OR
   iii. | Biwise OR
d. Shift Operators
   i. << shift Left
   ii. >>shift right
e. Identity operators
   i. Is is the identity same?
   ii. Is not is the identity not same?
f. Relational Operators
   i. <
   ii. >
   iii. <=
   iv. >=
   v. ==
   vi. !=
g. Logical Operators
   i. And Logical AND
   ii. Or Logical OR
h. Assignment operators
   i. =
   ii. /=
   iii. +=
   iv. -=
   v. %=
   vi. *=
   vii. **=
   viii. //=
i. Membership Operators
   i. In Whether variable in sequence
   ii. Not in whether variable is not in sequence.

E. Punctuators: Most commonly used punctuators in Python are:
‘ “ # \ () {} [] @ , : . =

Variable and Assignments:

A named memory location that refers to a value and whose value can be used and processed during program run is called variable.

Creating a variable:

```
Value = 50              # integer variable
Name = 'AMIT'           # String variable
Amount = 123.35         # Floating point variable
```

Note: Variables are not Storage Containers in Python. The variable in Python does not have fixed locations unlike other programming languages. The location they refer to change every time their values change.

Lvalues and Rvalues:

Lvalue: it is the expression that can comes on the lhs of an assignment.
Rvalue: It is the expression that comes on the rhs of the assignment.

Multiple assignments:

You can assign the same value to the multiple variables in the following way:
x = y = z = 20
Assigning multiple values to multiple variables:
x,y,z = 40,50,60

Dynamic Typing

If you assign a value to a variable and later another value of different data type you assign to the same variable, it doesn't give any error.

Simple Input and Output:

In Python, to get input from the user, an inbuilt function input () is used as follows:

Sname= input(" Students name:")
The input function always return the data of type string.

Reading number; As we know that input() returns string value therefore for numeric operations we need to convert these values into its int or float form using int() or float() function. For example:

amount = input("Enter amount")

Amt=float(amount)

Output through print Statement:

The syntax of using print() function of PYTHON is:
print(*Object, [sep = '' or <separator string> end = '\n' or <end-string.])

*Object means it can be one or more multiple comma separated object to be printed.
Example: print(22+5, "Year Old")

Exercise: Write questions and answers of you text book in class notes copy.

Chapter-4

Type-A: Short Answer Type Questions

Q.1	What are tokens in Python? How many types of tokens are allowed in Python? Exemplify your answer. Ans: The smallest individual unit of in a program is known as Token or a Lexical Unit. In Python the following types of tokens are allowed: (i) Keywords: Keywords are special meaning and it is reserved by the programming language. Python has the following keywords. Example – for, while, if, elif etc. (ii) Identifiers: Literals Literals are data items that have a fixed value. The types of literal are: • String: "INDIA" • Numeric: 56, 34.67 • Boolean: True, False • Special Literal: None (iii) Operators: a. Unary Operators b. Binary Operators c. Bitwise Operators etc. (iv) Punctuators: ' " # \ () {} [] @ , : . =
Q.2	How are keywords different from the identifiers?

	Ans: Keywords are reserve words used for special purpose, whereas identifiers are user defined words used to define variable, constants and objects etc.
Q.3	What are literals in Python? How many types of literals are allowed in Python? Ans: Literals Literals are data items that have a fixed value. The types of literal are: • String: "INDIA" • Numeric: 56, 34.67 • Boolean: True, False • Special Literal: None
Q.4	Can non-graphic characters be used in Python? How? Give examples to support your answer. Ans: Non-Graphic characters are those characters that cannot be typed directly from the keyboard e.g. backspace, tabs, carriage return etc. These characters are represented by escape sequence. Example: \n, \t, \v, \a etc.

2. Data Handling

This chapter includes more about data types, variables, operators and expressions used in Python.

Data Types

Data are used as integer, float and character or string type. Python offers following built in core data types:
(i)Number (ii) String (iii) List (iv) Tuple (v) Dictionary

(i) **Number:** The numbers in Python have following core data types:

a. *Integers*
- Integers Signed: Integers in Python can be of any length, it is only limited by the memory available. It is signed representation i.e. it can be positive as well as negative.
- Boolean: It represents false or true which behaves like 0 and 1. To get the Boolean equivalent of 0 or 1, you can type bool(0) or bool(1), Python will return false or true respectively.

b. *Floating Point:*
The Fractional numbers can be written in two forms:
- Fractional forms: Example: 5678.90, 34528.4532 etc.
- Exponent Form: 5.6789E03, 3.45284532E04

c. *Complex*: A complex number is a combination of real number and imaginary numbers. In complex number a+bi, a and b are real numbers whereas i is imaginary quantity which is represented by √-1. We can retrieve these two parts using attribute references. For complex number p: p.real will return real part of the complex number.
And p.imag will return the imaginary part as float, not as a complex number.

(ii) ***String***: A string data type lets you hold the string data which is any number of valid characters into a set of quotation mark.
Example: "pqrs", '$$$', "New Delhi", "3456.89" etc.
Forward and backward indexing of any string value can be represented in the following manner:

0	1	2	3	4	5	6	7	8	9	**Forward Indexing**
I	N	F	O	R	M	T	I	C	S	
										Backward Indexing

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

(iii) ***Lists and Tuples***:

These are compound data types of Python therefore we have taken together to study. But there is one difference, ***list can be modified but tuples cannot be modifies***.

A list in the python represents a list of comma-separated values of any data type between square brackets.

The following are some examples of List:

[1,3,5,7,9]

[“Amit”, “Sumit”, “Rajiv”, “Gaurav”]

[‘Namrata’, 202,404,302]

Tuples are represented as a list of comma separated values of any data types within parenthesis.

Example:

X= (1, 2, 3, 4, 5, 6, 7, 8)

Y= (‘a’, ‘e’, ‘i’, ‘o’, ‘u’)

List can be nested also. Example:

Head = [[‘A’, ‘B’, ‘C’, ‘D’, ‘E’,], [0,1,2,3,4,5,6,7]]

List = [‘x’, ‘y’, ‘z’]

List can be concatenated by using ‘+’ operator i.e.

```
>>> Head[1]+List would produce
[0,1,2,3,4,5,6,7,’x’, ‘y’, ‘z’]
```

To obtain the length of list following len command can be issue:

```
>>> print len(Head)
```

In addition, you can use the augmented addition assignment to add items to the list, but you must specify a list as the object to be added, as in the following example:

```
>>>List += [8]
```

(iv) ***Dictionary***:

The dictionary is the list of unordered set of comma separated value with the combination of **key: value** pair. It is declared within parenthesis {}. For example:

V = {‘a’:1, ‘e’:2, ‘i’:3, ‘o’:4, ‘u’:5}

Mutable and Immutable Types:

The data types used in Python are broadly classified into two categories: Modifiable (Mutable) and Non-modifiable (immutable).

1. **Immutable Types:**

   The immutable types are those that never change their values in place. Integer, float, Boolean, String and tuples come into this category.

   Example:

   X=8

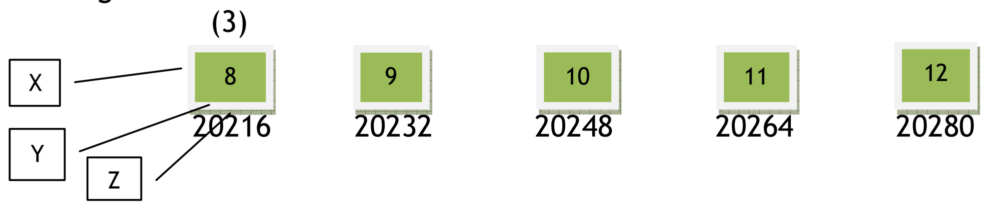
   Y=X

   Z=8

   It will produce 8,8,8

   In python values to the variables are stored differently in a different manner. Each value is having a fixed defined memory location. When we define any variable for any given value, the variable is referred to the location where the value is stored and as we change the value of the variable, the reference of the memory location for the variable is changed.

In Python it keeps a count internally to count that how many variables are referring a value.



You can check / confirm it yourself by using id(). It returns the memory address to which a variable is referencing.
In the above example

X=8
Y=X
Z=8

id(8)
id(X)
id(Y)
id(Z)
will give the same output because the values of all these identifiers are same.

2. **Mutable Types:**
   The mutable types are those whose values can be changed in place. These types are: **List, dictionaries and sets.**
   To change a member of a list, you may write:
   List1=[3,25,14]
   List1[1] = 80
   It will make the list namely List1 as [3,80,14]

Variable internal:
All data or values are referred to as object in Python.
In Python every object has three key attributes associated to it:
- (i) The type of an object
- (ii) The value of an object
- (iii) The id of an object

Variable names are stored as reference to a value – object. Each time you change the value, the variable's reference memory address changes.

Operator
1. **Arithmetic Operators**
   a. Unary Operators
   b. Binary Operators
   c. Augmented Assignment Operators
      x+=y => x=x+y
      x**=y => x = x**y
2. **Relational Operators**
   <, >, <=, >=, ==, !=
3. **Identity Operators (is, is not)**

Equality and Identity – Relation
is => returns true if both its operands are pointing to the same object
is not +> returns true if both operands are pointing to the different object.

4. **Logical Operators**
    a. OR operator
    b. AND operator
    c. NOT operator
5. **Bitwise Operators (&, |, ^, ~)**
    a. The AND operator and &
    b. The inclusive OR (|)
    c. The exclusive OR (^)
    d. The Complement Operator(~)

Expressions: Any valid combination of operators, literals and variables are known as expression. The expression in Python can be of any type:

1. Arithmetic Expressions
2. Logical Expressions
3. Relational Expressions
4. String Expressions

***Exercise*: Write questions and answers of you text book in class notes copy.**

3.Conditional Statements

Types of statements in Python:

1. Empty statement
2. Simple Statement
3. Compound statement

1. Empty Statement: A statement which does nothing is called empty statement in Python. Empty statement is pass statement. Whenever Python encounters a **pass** statement, Python does nothing and moves to the next statement in the flow of control.
2. Simple Statement: Any single executable statement is a simple statement. For example:
   Age = input (“Your Age?”)
   print(Age)
3. Compound Statement: It represents group of statements executed as a unit. It consists of **header** and **body.** The syntax is:
   <header>:
   <body containing multiple, simple or compound statement>

Statement Flow Control:

Programming statements are executed serially, selectively or iteratively.

Sequence: The execution of the statement follows the top to bottom approach.

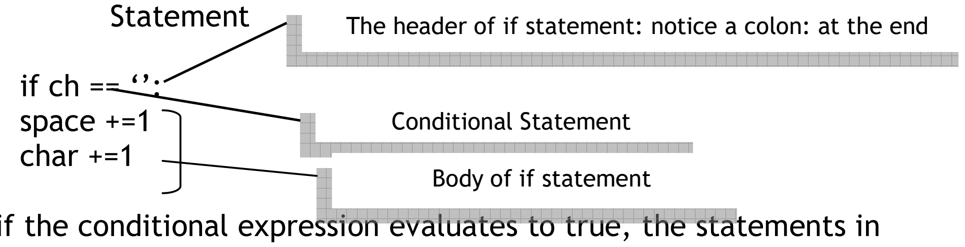
Selection: The execution of statement is based on the condition, if the condition evaluates to true, a group of statements are executed else other group of statements are executed.

Iteration (Looping): When the set of statements are executed repeatedly based on a condition, it is called iteration statement.

The *if* Statement of Python:

The if statement in Python constructs the selection statement. Its syntax is as shown below:

```
if<Conditional Expression>:
        Statement
```



```
if ch == ‘’:
space +=1
char +=1
```

Here in this case, if the conditional expression evaluates to true, the statements in the body of if are executed, otherwise ignored.

Example:

```
X = int(input("Enter the value of x:"))
Y = int(input("Enter the value of y:"))
 if A>20 and B<50:
        z=(x-y)*y;
        print("The result is:",z)
print("Thank you")
```

This statement is not the part of if as it is not indented at the same level as that of body of if statement.

<u>The if - else statement:</u>

This type of if statement test a condition and if the condition evaluates to true the body of if will be executed, otherwise else part will be executed.
Its Syntax is:

```
        if <Condition>:
                statement
        else
                statement
```

For example:

```
        if age>=18:
        print("You are eligible to get driving license")
        else
        print("You are not eligible")
print("Thank You")
```

Example: Program to check whether a given number is odd or even.

```
        N1 = int(input("Enter an integer:")
        if N1%2==0:
                print(N1, "is an Even Number")
        else :
                print(N1, "is an Odd Number")
```

<u>Multiple if..elif</u>

Syntax:

```
        if <condition1>:
                Statement1
        elif<condition2>:
                Statement2
        elif<condition3>:
                Statement3
        else:
                Statement4
```

The nested if statement:

If an if condition is tested within another if condition then it is called nested if.
The syntax is(Nested within if):

```
          if <conditional expression1>:
                If<conditional expression2>:
                        Statements
                else:
                        Statements
          elif<conditional expression3>:
                statements
```

```
        else:
                statements
```

Another Syntax(Nested within elif):

```
        if <conditional expression1>:
                Statements
        elif<conditional expression2>:
                If<conditional expression3>:
                        Statements
                else:
                        statements
        else:
                statements
```

Related questions:
# Program to read three numbers and print them in ascending order.
#ABC shop deals in apparels and footwear. Write a program to calculate total selling price after levying the GDT. Do calculate central govt. GST and state govt. GST rates as applicable below:

Item	GST rate
Footwear<=500 (per pair)	5%
Footwear >500(per pair)	18%
Apparels <=1000(per piece)	5%
Apparels >=1000(per piece)	12%

Storing conditions:

The complex conditions can be stored under the name and used further, for example:

If percentage of marks is more than 90% and marks in maths is more than 80%, admission confirms in Maths stream.
If percentage of marks is more than 80% and marks in science is more than 80%, admission confirms in Bilogy stream.
If percentage of marks is more than 70% and marks in SocSc is more than 80%, admission confirms in Commerce stream.
Else admission closed for below 70% marks

The above statements can be written as follows:

```
MAths = per>=90 and Marks_Maths>80
Sc = per>=80 and Marks_Sc>80
Com=per>=70 and Marks_SSc>80
```

Now you can use these named conditionals in your coding as follows:

```
        if Maths:
                Stream=”Mathematics”
        elif Sc:
```

```
        Srtream= “Science”
    elif Com:
        Stream = “Commerce”
    else :
        Stream = “No admission”
```

Formatting output: Output can be produced systematically by using format(). The syntax is:

“<Text>{<data Parameter>:<Specified Format for data>}”.format(Value)

Example:

“Marks:{0:f}”. format(45)
“Marks:{0:f}   Percent{1:5.2f}”. format(65.98)

<u>Repetition of Tasks:</u>

Before learning the looping concept in PYTHON we should know the following range() function. The range() function of PYTHON generates a list which is a special sequence type. A sequence in PYTHON is a succession of values bound together by a single name. Some PYTHON sequence types are : string, list, tuples etc.

The syntax of range() function is:
range(<lower limit>,<upper limit>)   #both limit should be integers
Example: range(0,6) will produce a list as [0,1,2,3,4,5].
range(6,0) will return an empty list []
To produce a list with numbers having gap other than 1, then the syntax of range would be:
range(<lower limit>,<upper limit>,<step value>)
Example: range(0,8,2) will produce a list as [0,2,4,6]
range(6,0,-1) will produce a list as [6,5,4,3,2,1]

Another form of range is range(number) which produce a list from number 0 to the specified number in the range.
For example range(6) will produce a list as [0,1,2,3,4,5]

in and ***not in*** Operators in PYTHON:

These operators are used with range() function in for loop. To check whether a value is contained inside a list you can use in operator i.e.

3 in [0,1,2,3,4,5,6] will return True and 8 in [0,1,2,3,4,5,6] will return False.

Whereas 8 not in [0,1,2,3,4,5,6] will return True value.

These operators work with all sequence types i.e. string, tuples, list etc.

For example:
‘S’ in “Strength” will return True

Now consider the following code that uses in operator:

```
line = input("Enter a line:")
string=input("Enter a string:")

if string in line:
        print(string, "is part of", line)
else:
        print(string, "is not the part of", line)
```

<u>Iterative Statements</u>: These statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. PYTHON provides two types of loops: for loop and while loop. These loops represents two categories of loops which are:

#Counting loops: The loop that repeat a certain number of times. Ex. *for* loop.

#conditional loop: The loop that repeat until a certain thing happens i.e. they keep repeating as long as some condition is true. Ex. *while* loop.

<u>The for loop</u>:

Syntax:

```
        for <variable> in <sequence>:
                statement to repeat
```

Example:

```
        for a in[1,4,7]:
                print(a)
                print(a*a)
```

Example: Printing multiplication table of a given number.

```
num=5
for a in range(1,11):
        print(num, 'x', a, '=',num*a)
```

The above code will generate the multiplication table of 5.

Example: Program to print sum of natural numbers between 1 and 10.

```
sum=0
for n in range(1,11):
        sum+=n
        print("The sum of first 10 natural number is", sum)
```

The ***<u>while</u>*** loop:

Syntax:

```
        while <logical expression>:
                loop-body
```

Example:

```
        a=5
        while a>0:
                print(“Hello”,a)
                a=a-3
        print(“Loop Over!”)
```

The Loop else Statement:
The else clause of loop executes when the loop terminates normally, it does not execute when the loop is terminated through break statement.
For example:

```
for x in range (1,6):
        if x%2==0:
                break
        print(“The number is =:”, end=’’)
        print(a)
else:
        print(“All numbers are printed”)
```

It works with the while loop also:

```
c=s=0
ans=’y’

while ans==’y’:
        num=int(input(“Enter a number:”))
        if num<0:
                print(“Negative numbers are not accepted: Exiting!!!”)
                break
        s+=num
        c+=1
        ans=input(“Would you like to enter more number?(y/n)”)
else:
        print(“You have entered”, c, “numbers”)
print(“The total is”, s)
```

Jump Statements:

break and continue are used in python as Jump statements.
The break statement skips the rest of the loop and jumps out of the loop statement whereas continue statement skips only that part of the loop but remain in the loop and executed.

Example:

```
for  x in range(1,11):
        if x%5==0:
                break
        print(“The number is:”, x)
```

The output would be:

```
1
2
3
4
```

```
for  x in range(1,11):
        if x%5==0:
                continue
        print(“The number is:”, x)
```

The output would be:

```
1
2
3
4
6
7
8
9
10
```

Nested Loops

If a loop is contained inside another loop then it is called nested of loops.

```
for  x in range(1,4):
        for y in range(1,x):
                print(y, end= ‘’)
        print()
```

Exercise: **Write questions and answers of the chapter 6 in class notes copy.**

4. Text Handling

Objectives:
#Traverse a String
#String Operators
#String Slices
#String functions and Methods

Traverse a String:
Traverse refers to iterative through the elements of string one character at a time.

Example:

```
S= “OPJS”
for c in S:
        print(c, ‘$’,end = ‘’)
```

it will print:

O$P$J$S$

String operator

Concatenation Operator ‘+’

This operator ‘+’ is used to combine two strings.

Example:
S1= “INFORMATICS”
S2= “PRACTICES”
S=S1+S2
Print(S)

Output: INFORMATICSPRACTICES

4+6 = 10 (Numeric addition)
4+ “6” = Invalid
“4”+ “6” = 46 (String concatenation)

Replication Operator ‘*’
It needs two operands string and number, the string operand tells the string to be replicated and number operand tells the number of times, it is to be replicated.

Example:

Input	Output
“”xy”*3	xyxyxy
3* “pq”	pqpqpq
“$”*4	$$$$
“4”* “2”	Invalid

Membership Operators

in and not in are called membership operators in python. They return Boolean True or False value. Examples are:

“A” in “AJAY”	will return True
“OP” in “OPJS”	Will return True
“OP” not in “India”	Will return True
“X” not in “XYZ”	Will return False
“x” in “XYZ”	Will return False

Comparison Operators

The comparison operators <, >, <=, >=, ==, != are also applied to string in Python. The comparison is based on character to character.

Examples:

“x”== “X”	False
“x”!= “X”	True
“pqr”!= “PQR”	True
‘p’ < ‘P’	False
‘ÁBC’ > ‘AB’	True
‘abcd’ > ‘abcD’	True

<u>Determining Ordinal value of a single character:</u>

The ordinal or Unicode value of a character can be determined by ord() function. Example:

ord(‘B’) will return 66

The opposite of ord() function is chr(), which is used to convert a numeric value into an equivalent character. For example:

Chr(67) will return C

<u>String Slices(Substring)</u>

String slice means a part of the string, part of a string is obtained in the following manner:

The word ‘INFORMATICS’

0	1	2	3	4	5	6	7	8	9	**Forward Indexing**
I	N	F	O	R	M	T	I	C	S	
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	**Backward Indexing**

word[0:10]	INFORMATICS
word[0:4]	INFO
word[2:5]	FOR
word[-8:-4]	FORM
word[:10]	INFORMATICS
word[6:]	TICS
word[1:6:2]	NOM
word[-7:-3:3]	OT
word[::-2]	SIMON
word[::-1]	SCITAMROFNI

The len() function: To obtain length of the string. Here length of the string means total number of characters a string has.

Example:

```
Str1= “OPJS RAIGARH”
Length=len(str1)
Print(Length)
```

```
Output:12
```

```
Str2 = Str1[1:5]
Length =len(Str2)
Print(Length)
```

```
Output:4
```

Exercise: Questions and Answers of Chapter 7

5. List Manipulation

Objectives
#Concept of List
#Creating a List
#Accessing List items
#Joining Lists
#Repeating List
#Slicing the List
#Appending Elements to a List
#Updating Element to a List
#Deleting Elements from a List
#List Functions and Methods

List is a standard data type in Python which contains the values of any data types. Python list is mutable. List is depicted through pair of square brackets [].
Example:

[]	Empty list
[1,2,3]	list of integers
[‘kamlesh’, ‘isha’, ‘malti’]	list of students
[1, ‘kamlesh’,5.5, ‘malti’]	List of mix data values

Creating a list:
A list can be any value separated by comma and kept inside the pair of square brackets.

To create a list we can use the following statement:

```
L1 = [‘kamlesh’, ‘isha’, ‘malti’]
L2 = [1,2,3]
```

This is known as list display construct.

Types of Lists:

- Empty List
  If a list doesn't contain any item, it is called empty list
  The empty list is equivalent to 0 or '' and its truth value is false.
  We can also create and empty list as follows:
  L3=list()
- Long List

A list can be separated in many lines due to long list of items. Such list is known as Long List.

```
Values = [2,4,6,8,90,23,34,56,78,21,33,53,14,15,
          45,67,56,35,25,78,34,78,24,78,22,57]
```

- Nested List

A list can have a list as an element, which is known as nested list.
List1 = [4,56,[23.70,45,50],78]

Creating a list from Existing Sequence:
The built in list type object can also be used to create a list. For example:

List1=list(<sequence>)
Where the sequence can be any types of sequence object including strings, tuples and lists. Python creates individual elements of the list from the individual elements of passed sequence. If you pass in another list, the list function makes a copy. For Example:

```
L=( 'Q', 'W', 'E', 'R', 'T', 'Y')
L1=list(L)
L1
['Q', 'W', 'E', 'R', 'T', 'Y']
```

This method can be used to create lists of single character or single digit through keyboard input also.

Example:

```
List = list(input( "Please enter the list element:"))
Please enter the list element: asdfgh
>>>List
['a', 's', 'd', 'f', 'g', 'h']

Please enter the list element: 567890
>>>List
['5', '6', '7', '8', '9', '0']
```

Here the data types of all the elements are string even though we had entered numeric values.
 To enter a list of integers, we can use the following method:

```
List =eval(input("Enter list to be added:")
Print("List you entered:", List)

Enter list to be added: 567890
List you entered:[5,6,7,8,9,0]
```

Sometime the function eval() does not work in Python shell. We can run it through script. The eval() function of Python can be used to evaluate and return the result of an expression given as String. For Example:

Eval(‘4+9’) will give us a result as 13
Similarly
X= eval(“4*10”)

Print(X) will produce 40

V1=eval(input(“Enter Value:”))
Print(v1)

Enter Value: 22+6
28

Enter Value: [3,4,5]
[3,4,5]

Accessing Lists:

List elements are stored like string in Python. They also have the index like string. The list name stores references of where its elements are stored index-wise. Each of the individual items of the list are stored somewhere else in the memory. To access the list we have to follow the same methodology as we did for string. List[i] will extract the element at ith index position of the list.

List are similar to string in the following ways:
Length - To returns the number of items.
Index and slicing – List[i] and List[i:j] are the ways to extract the list items.
Membership Operators- ‘in’ and ‘not in’ operators work like they work for string.
Concatenation and Replication operators ‘+’ and ‘*’ - The ‘+’ adds one list to another whereas ‘*’ operator repeats the list.

Differences from the string:
Strings are not mutable whereas Lists are mutable.

Example: vovels=[‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

vovels[0] = ‘A’ will give the updated elements of a list as follows:

[‘A’, ‘e’, ‘i’, ‘o’, ‘u’]

Traversing a List:

```
for a in vovels:
        print(a)
```
It will produce result like:
```
a
e
i
```

o
u

$\longleftrightarrow$

Comparing List
Lists can be compared by using '==', '<', '>', '<=', '>=' etc. Lists elements are compared based on their index positions. For examples:

[3,4,5,6,7] < [5,6,7] will return true because 3<5
[3,4,5,6,7] < [3,4,9,7] will also return true because 5<9, but
[3,4,5,6,7] < [3,4,5,2,7] will return false as 6 is not less than 2.

List Operations:

Joining the Lists:
L1=[2,3,4]
L2=[6,7,8]
L1+L2 will give the result [2,3,4,6,7,8]. The '+' operator will work only for list type element any other data type can not be added into the list.
List+number, List+String will produce an error.
Repeating a list:
L1*3 will produce [2,3,4,2,3,4,2,3,4]

Slicing the list:
Sub part of a list can be extracted through list slice.
For example:

```
L1=[1,2,3,4,5,6,7,8,9]
>>> L1=[1,2,3,4,5,6,7,8,9]
>>> ext = L1[3:-3]
>>> ext
[4, 5, 6]
>>> ext = L1[3:-4]
>>> ext
[4, 5]
>>> ext = L1[3:-5]
>>> ext
[4]
```

In the slicing of the list items in Python, it simply returns the elements of the list which fall between the boundaries, if any, without raising any error.

For example:

```
>>> list1=[2,3,4,5,6,7,8,9,10,11,12,13,14]
>>> list1[3:30]
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> list1[-12:5]
[3, 4, 5, 6]
>>> list1[2:15]
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> list1[-15:15]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
>>> list1[15:30]
[]
```

We can also extract the alternate values in the list by using the following format:
R=list[start:stop:steps]

```
>>> list1[2:10:3]
[4, 7, 10]
>>> list1[2:10:2]
[4, 6, 8, 10]
>>> list1[2:10:4]
[4, 8]
>>> list[:3:2]
>>> list1[:3:2]
[2, 4]
>>> list1[::3]
[2, 5, 8, 11, 14]

>>> list1[5::2]
[7, 9, 11, 13]
```

To reversing the list

```
>>> list1[::-1]
[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
```

Working with Lists:

Appending Elements to a list by using the format:
List.append(item)

```
>>> L1=[1,2,3,4,5,6,7,8]
>>> L1.append(12)
>>> L1
[1, 2, 3, 4, 5, 6, 7, 8, 12]
```

Updating elements to a list by using the format:
List[index] = <new value>

```
>>> L1
[1, 2, 3, 4, 5, 6, 7, 8, 12]
>>> L1[4]=14
>>> L1
[1, 2, 3, 4, 14, 6, 7, 8, 12]
```

Deleting elements from a list by using the format:
del list[index]
del list[start:stop]

```
>>> list1
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> del list1[5]
>>> list1
[2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14]
>>> del list1[2:6]
>>> list1
[2, 3, 9, 10, 11, 12, 13, 14]
>>> del list1  # to remove a list completely
```

<u>The pop() method</u>:
It removes an element from the list and returns it. It is the same as del but pop method also returns the deleted element along with the list.

The syntax is:
List.pop(index)
The pop method is useful only when you want to store the element being deleting for later use.

```
>>> L1
[1, 2, 3, 4, 14, 6, 7, 8, 12]
>>> L1.pop()  ## removing the last item.
12
>>> L1.pop(5) ## removing the 6th
6
```

To store the deleted item from the list we can use the following:

```
X= L1.pop()   # X will store the value 12
Y=L1.pop(5)  # Y will store the value 6
```

<u>Making True copy of a List</u>:

```
>>> L1
[1, 2, 3, 4, 14, 7, 8]
>>> L2=L1              # Copying L1 to L2
>>> L1[2]=5            # Updating record of L1
>>> L1
[1, 2, 5, 4, 14, 7, 8]
>>> L2                 # Changes reflected in L2 also
[1, 2, 5, 4, 14, 7, 8]
```

<u>List functions and methods</u>:
Every list object is an instance of List class. Therefore list manipulation methods can also be applied on list objects in the following format:
<List Object>.<method name>()

1. The index method: This method returns the index of first matched item from the list. The syntax is:
List.index(item)

```
>>> L1
[1, 2, 5, 4, 14, 7, 8]
>>> L1.index(4)              # index value of item 4
3
>>> L1.index(5)
2                            # index value of item 5
```

2. The append method: Adds the item to the end of the list. The syntax is:

List.append(item)

```
>>> name=['Manish', 'vasudha','gurmeet','neha']
>>> name.append('Amit')
>>> name
['Manish', 'vasudha', 'gurmeet', 'neha', 'Amit']
```

3. The extend method: It is used to add multiple elements given in the form of list. The syntax is:

List.extend(list)

```
>>> name
['Manish', 'vasudha', 'gurmeet', 'neha', 'Amit']
>>> title=['Sharma','Kumar','Murthy']
>>> name.extend(title)
>>> name
['Manish', 'vasudha', 'gurmeet', 'neha', 'Amit', 'Sharma', 'Kumar', 'Murthy']
                                                   # List is extended
>>> title                                          # No change in appended
['Sharma', 'Kumar', 'Murthy']
```

Difference between append() and extend() method
The append method adds one element to a list extend() method add multiple element from a list supplied to it as an argument. Consider the following examples:

```
>>> L1
[1, 2, 5, 4, 14, 7, 8]
>>> L1.append(16)
>>> L1
[1, 2, 5, 4, 14, 7, 8, 16]          # A single item has been appended
>>> L1.append(20,21)
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    L1.append(20,21)                # Two items cannot be appended
TypeError: append() takes exactly one argument (2 given)
>>> L1.append([20,21])
>>> L1
[1, 2, 5, 4, 14, 7, 8, 16, [20, 21]]
                                    # But items in the form of a list can be appended.

>>> L1
[1, 2, 5, 4, 14, 7, 8, 16, [20, 21]]
>>> L1.extend(13)                   # Single element cannot be extended.
            Traceback (most recent call last):
              File "<pyshell#62>", line 1, in <module>
                L1.extend(13)
            TypeError: 'int' object is not iterable
>>> L1.extend([18,20])              # Elements in the form of List can be extended.
>>> L1
[1, 2, 5, 4, 14, 7, 8, 16, [20, 21], 18, 20]
>>> L1 # But a single element kept inside the [] can be used for extension of a list.
[1, 2, 5, 4, 14, 7, 8, 16, [20, 21], 18, 20, 34]
```

6. The insert method:

This method is used to insert an item within the list. The syntax is:

```
List.insert(<pos>,<item>) # It does not return any value.
```

Example:

```
>>> L1=[2,3,4,5,6]
>>> L1.insert(2,23)
>>> L1
[2, 3, 23, 4, 5, 6]

>>> L1.insert(0,50)
>>> L1
[50, 2, 3, 23, 4, 5, 6]
>>> L1.insert(len(L1),40)
>>> L1
[50, 2, 3, 23, 4, 5, 6, 40]
>>> L1.insert(-1,80)          #This will not add the item at last because the position
>>> L1                        argument take the index value before the item has to be
[50, 2, 3, 23, 4, 5, 6, 80, 40]     inserted.
```

7. The pop() method

This method is used to remove an element from the list. The syntax is:

```
List.pop(<index>)
```

Example:

```
>>> L1.pop(0)                 # To remove first element
50
>>> L1
[2, 3, 23, 4, 5, 6, 80, 40]
>>> L1.pop()                  # To remove the last element
40
>>> L1
[2, 3, 23, 4, 5, 6, 80]

>>> L1.pop(2)                 # To remove the element at specified index position.
23
>>> L1
[2, 3, 4, 5, 6, 80]
```

8. The remove() method:

The pop() is used to remove the element based on its index position, but if you don't know the index position then remove() method will remove the first occurrence of the element. For example:

```
>>> L1
[2, 3, 4, 5, 6, 80, 6, 3, 4, 5, 12]
>>> L1.remove(4)            # It removes the first occurrence of 4
>>> L1
[2, 3, 5, 6, 80, 6, 3, 4, 5, 12]
```

9. The clear() method:

To remove all the items from the list this method is used. The syntax is:
List.clear()

For Example:

```
>>> L1
[2, 3, 5, 6, 80, 6, 3, 4, 5, 12]
>>> L1.clear()
>>> L1
[]
```

10. The count() method:

It returns the count of the item that a user passed as argument. If the given item is not in the list it returns the zero. The syntax is:
List.count(<item>)

For example:

```
>>> L1=[2, 3, 5, 6, 80, 6, 3, 4, 5, 12]
>>> L1.count(6)
2
>>> L1.count(5)
2
```

11. The reverse() method: It reverses the items of the list. The syntax is:

List.reverse()

For Example:

```
>>> L1
[2, 3, 5, 6, 80, 6, 3, 4, 5, 12]
>>> L1.reverse()
>>> L1
[12, 5, 4, 3, 6, 80, 6, 5, 3, 2]
```

12. The sort() method: It arranges the list items in increasing or decreasing order. By default it arranges the items in increasing order. The syntax is:

```
List.sort()                 # For increasing order(By default)
List.sort(reverse=True)     # For decreasing order
```

```
>>> L1
[12, 5, 4, 3, 6, 80, 6, 5, 3, 2]
>>> L1.sort()
>>> L1
[2, 3, 3, 4, 5, 5, 6, 6, 12, 80]
>>> L1.sort(reverse=True)
>>> L1
[80, 12, 6, 6, 5, 5, 4, 3, 3, 2]
```

Exercise (Questions and Answers)

Type: A

1.Discuss the utility and significance of Lists, briefly.
Ans: List is used to store values of any type. List is mutable i.e. any value of the list can be changed at its own place.

2. What do you understand by mutability? What does "in place" memory updation means?
Ans: Lists are mutable i.e. you can change elements of a list in place. In other words, the memory address of a list will not change even after you change its values. The value of the list is updated on its own place i.e. index position.

3. Start with the list [8,9,10]. Do the following using list function.
a. Set the second entry to 17.
Ans:

```
>>> List=[8,9,10]
>>> List[1] = 17
>>> List
[8, 17, 10]
```

b. Add 4,5,6 to the end of the list.

```
>>> List.extend([4,5,6])
>>> List
[8, 17, 10, 4, 5, 6]
```

c. Remove the first entry from the list.

```
>>> List
[8, 17, 10, 4, 5, 6]
>>> del List[0]
>>> List
[17, 10, 4, 5, 6]
```

OR

```
>>> List.pop(0)
17
>>> List
[10, 4, 5, 6]
```

d. Sort the list.

```
>>> List
[10, 4, 5, 6]
>>> List.sort()
>>> List
[4, 5, 6, 10]
```

e. Double the list.

```
>>> List
[4, 5, 6, 10]
>>> List*2
```

```
[4, 5, 6, 10, 4, 5, 6, 10]
```

f. Insert 25 at index 3.

```
>>> List.insert(3,25)
>>> List
[4, 5, 6, 25, 10]
```

4. If a is [1,2,3]
   a. What is the difference(if any) between a*3 and [a,a,a]
      ```
      >>> a=[1,2,3]
      >>> a*3
      [1, 2, 3, 1, 2, 3, 1, 2, 3]  # List elements are displayed three times.
      >>> [a,a,a]
      [[1, 2, 3], [1, 2, 3], [1, 2, 3]]       # List elements are nested.
      ```
   b. Is a*3 equivalent to a+a+a?
      Yes, these are equal.
      ```
      >>> a*3
      [1, 2, 3, 1, 2, 3, 1, 2, 3]
      >>> a+a+a
      [1, 2, 3, 1, 2, 3, 1, 2, 3]
      ```
   c. What is the meaning of a[1:1] = 9?
      It produces the following error. Since a[1:1] returns an empty list, therefore the value 9 is not assigned to the empty list.
      ```
      >>> a[1:1]=9
      Traceback (most recent call last):
        File "<pyshell#20>", line 1, in <module>
          a[1:1]=9
      TypeError: can only assign an iterable
      >>> a[1:1]
      []
      ```
   d. What is the difference between a [1:2] =4 and a [1:1] =4?
      The a[1:2] = 4 and a [1:1] both expression will produce an error. To use a[1:2] =4, 0 we get the following output.
      ```
      >>> a[1:2]=4,0
      >>> a
      [1, 4, 0, 3]    # it is replacing the list value 2 with 4 and 0.
      ```
5. What is a[1:1] if a is string of at least two characters? And what if string is shorter?
   Both will return the blank string. For example:
   ```
   >>> a = 'op'
   >>> a[1:1]
   ''
   >>> a = 'p'
   >>> a[1:1]
   ''
   ```
6. What is the purpose of del operator and pop method? Try deleting a slice.
   del is used to Delete elements from a list by using the format:
   del list[index]
   del list[start: stop]

```
>>> list1
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> del list1[5]
>>> list1
[2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14]
>>> del list1[2:6]
>>> list1
[2, 3, 9, 10, 11, 12, 13, 14]
>>> del list1  # to remove a list completely
```

The pop() method:
It removes an element from the list and returns it. It is the same as del but pop method also returns the deleted element along with the list.

The syntax is:
List.pop(index)
The pop method is useful only when you want to store the element being deleting for later use.

```
>>> L1
[1, 2, 3, 4, 14, 6, 7, 8, 12]
>>> L1.pop()  ## removing the last item.
12
>>> L1.pop(5) ## removing the 6th
```

7. What does each of the following expressions evaluate to? Suppose that L is the list.
   [“These”,[“are”, “a”, “few”, “words”], “that”, “we”, “will”, “use”]
   a. L[1][0::2]
      ```
      >>> L=["These", ["are", "a", "few", "words"], "that", "we", "will", "use"]
      >>> L[1][0::2]
      ['are', 'few']
      ```
   b. “a” in L[1][0]
      ```
      >>> "a" in L [1][0]
      True
      ```
   c. L[:1]+L[1]
      ```
      >>> L[:1]+L[1]
      ['These', 'are', 'a', 'few', 'words']
      ```
   d. L[2::2]
      ```
      >>> L[2::2]
      [‘That', 'will']
      ```
   e. L[2][2] in L[1]
      ```
      >>> L[2][2] in L[1]
      True
      ```

8. What are the List Slices? What for can you use them?
   Slicing the list:

Sub part of a list can be extracted through list slice.
For example:

```
L1= [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> L1= [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> Ext = L1 [3:-3]
>>> Ext
[4, 5, 6]
```

In the slicing of the list items in Python, it simply returns the elements of the list which fall between the boundaries.
We can also extract the alternate values in the list by using the following format:

```
R=list [start: stop: steps]
>>> list1=[2,3,4,5,6,7,8,9,10,11,12,13,14]
>>> list1 [2:10:3]
[4, 7, 10]
>>> list1 [2:10:2]
[4, 6, 8, 10]
```

9. Does the slice operator always produce a new list?
Yes, it produces a new list as it is extracted from the original list. For example:

```
>>> list1=[2,3,4,5,6,7,8,9,10,11,12,13,14]
>>> list1 [2:10:3]
[4, 7, 10]
```

Another example:

```
>>> L1= [1,2,3,4,5,6]
>>> L2=L1 [2:5]
>>> L2
[3, 4, 5]
```

10. Compare list with strings. How are they similar and how are they different?
Similarities:
a. Indexing:

>>> s="informatics"
>>> s[3]
'o'
>>> s[5]
'm'
	>>> L1=[1,2,3,4,5]
	>>> L1[3]
	4

b. Slicing :

word[0:10]	INFORMATICS	L1=[1,2,3,4,5,6,7,8,9]
word[0:4]	INFO	>>> L1=[1,2,3,4,5,6,7,8,9]
word[2:5]	FOR	>>> ext = L1[3:-3]
word[-8:-4]	FORM	>>> ext
		[4, 5, 6]

c. Reversing:

>>> s[::-1] 'scitamrofni'	>>> L1[::-1] [5, 4, 3, 2, 1]

Differences:

List is Mutable means without creating a new object list value can be changed on its own position where as string is immutable.

11. What do you understand by true copy of a list? How it is different from shallow copy?
Ans: A true copy of the list can be created by using list() method in the following manner:
```
>>> L1
[1, 2, 3, 4, 5]
>>> L2=list (L1)
>>> L2
[1, 2, 3, 4, 5]
```

Both the lists are independent of each other i.e. L2 is remain unchanged if L1 is modified.

If it could be created as:
```
>>> L2=L1
>>> L2
[1, 2, 3, 4, 5]
>>> L1[2]=10
>>> L1
[1, 2, 10, 4, 5]
>>> L2
[1, 2, 10, 4, 5]
```
Here change is reflected in L2 as L1 is updated. This is called shallow copy of the list.

12. An index out of bound given with a list name causes error, but not with list slice. Why?
Ans: The index out of bound **means** you are providing an **index** for which a **list** element **does** not exist. E.g, if your **list** was [1, 3, 5, 7] , and you asked for the element at **index** 10, you**would** be well **out of bounds** and receive an error, as only elements 0 through 3 exist.

Type: B

1. What is the difference between following two expressions, if lst is given as [1,3,5]
   (i) lst*3
   Ans: The lst*3 will give the following output:
   ```
   >>> lst=[1,3,5]
   >>> lst*3
   [1, 3, 5, 1, 3, 5, 1, 3, 5]
   ```
   but the elements of lst will remain same i.e. [1,3,5]

   (ii) lst*=3
   Ans: Whereas lst*=3 will be equal to lst=lst*3 which will change the contents of lst and the contents would be:
   [1, 3, 5, 1, 3, 5, 1, 3, 5]

2. Given two lists
L1=[“this”, “is”, “a”, “List”]   L2=[“This”, [“is”, “another”], “List”]

Which of the following expressions will cause an error and why?

(a) L1==L2
(b) L1.upper()
(c) L1[3].upper()
(d) L2.upper()
(e) L2[1].upper()
(f) L2[1][1].upper()

```
>>> L1=['this','is', 'a','List']
>>> L2=['this',['is','another'],'List']
```

Ans:

```
 (a) >>> L1==L2
     False
 (b) >>> L1.upper()
     AttributeError: 'list' object has no attribute 'upper'
 (c) >>> L1[3].upper()
     'LIST'
(d)>>> L2.upper()
    AttributeError: 'list' object has no attribute 'upper'
 (e) >>> L2[1].upper()
    AttributeError: 'list' object has no attribute 'upper'
 (f) >>> L2[1][1].upper()
    'ANOTHER'
```

3. From the previous question, give the output of the expressions that do not result in error:

Ans:

```
(a) >>> L1==L2
        False
(b)>>> L1[3].upper()
        'LIST'
(c)>>> L2[1][1].upper()
        'ANOTHER'
```

4. Given a list L1=[3,4.5,12,25.7,[2,1,0,5],88]

Ans:

(a) Which List slice will return [12,25.7,[2,1,0,5]]

```
>>> L1=[3,4.5,12,25.7,[2,1,0,5],88]
>>> L1[2:5]
    [12, 25.7, [2, 1, 0, 5]]
```

(b) Which expression will return [2,1,0,5]

```
>>> L1[4]
    [2, 1, 0, 5]
```

(c) Which list slice will return [[2,1,0,5]]

```
>>> L1[4:5]
    [[2, 1, 0, 5]]
```

(d) Which list slice will return[4.5,25.7,88]

```
>>> L1[1:6:2]
    [4.5, 25.7, 88]
```

5. Given a list L1=[3,4.5,12,25.7,[2,1,0,5],88] , which function can change the list to:
   Ans:
   (a) [3,4.5,12,25.7,88]
   ```
   >>> L1[0:4]+L1[5:6]
       [3, 4.5, 12, 25.7, 88]
   ```
   (b) [3,4.5,12,25.7]
   ```
   >>> L1[0:4]
       [3, 4.5, 12, 25.7]
   ```
   (c) [[2,1,0,5],88]
   ```
   >>> L1[4:6]
       [[2, 1, 0, 5], 88]
   ```
6. What will the following code result in?
   ```
   L1=[1,3,5,7,9]
   print(L1==L1.reverse())
   print(L1)
   ```
   Ans:
   ```
   >>> L1=[1,3,5,7,9]
   >>> print(L1==L1.reverse())
           False
   >>> print(L1)
           [9, 7, 5, 3, 1]
   ```
7. Predict the output:
   ```
   my_list=[‘p’, ‘r’, ‘o’, ‘b’, ‘l’, ‘e’, ‘m’]
   my_list[2:3]=[]
   print(my_list)
   my_list[2:5]=[]
   print(my_list)
   ```
   Ans:
   ```
   >>> my_list=['p','r','o','b','l','e','m']
   >>> my_list[2:3]=[]
   >>> print(my_list)
           ['p', 'r', 'b', 'l', 'e', 'm']
   >>> my_list[2:5]=[]
   >>> print(my_list)
           ['p', 'r', 'm']
   ```
8. Predict the output
   ```
   List1=[13,18,11,16,13,18,13]
   print(List1.index(18))
   print(List1.count(18))
   List1.append(List1.count13))
   print(List1)
   ```
   Ans:
   ```
   >>> List1=[13,18,11,16,13,18,13]
   >>> print(List1.index(18))
           1
   >>> print(List1.count(18))
           2
   >>> List1.append(List1.count(13))
   ```

```
>>> print(List1)
        [13, 18, 11, 16, 13, 18, 13, 3]
```

9. Predict the output:

```
Odd=[1,3,5]
print((Odd+[2,4,6])[4])
print((Odd+[12,14,16])[4]-(Odd+[2,4,6])[4])
```

Ans:

```
>>> Odd=[1,3,5]
>>> print((Odd+[2,4,6])[4])
        4
>>> print((Odd+[12,14,16])[4]-(Odd+[2,4,6])[4])
        10
```

10. Predict the output:

```
a,b,c =[1,2],[1,2],[1,2]
print(a==b)
print(a is b)
```

Ans:

```
>>> a,b,c=[1,2],[1,2],[1,2]
>>> print(a==b)
        True
>>> print(a is b)  #
        False
```

11. Predict the output of the following two parts. Are the outputs same? Are the output different?

(a)
```
L1,L2=[2,4],[2,4]
L3=L2
L2[1]=5
print(L3)
```

Ans:

```
>>> L1,L2=[2,4],[2,4]
>>> L3=L2
>>> L2[1]=5
>>> print(L3)
[2, 5]      # Change is reflected in another list True copy of the List
```

(b)
```
L1,L2=[2,4],[2,4]
L3=list(L2)
L2[1]=5
print(L3)
```

Ans:

```
>>> L1,L2=[2,4],[2,4]
>>> L3=list(L2)
>>> L2[1]=5
>>> print(L3)

[2, 4]      # No change is reflected in the list
```

12. Find the errors:

Ans:

a. L1=[1,11,21,31]

b. L2=L1+2 # error:- anything other than a list is added
c. L3=L1*2
d. Idx=L1.index(45) #error:- 45 is not in the list

13. Find the errors:

Ans:

```
(a) L1=[1,11,21,31]
    An=L1.remove(41)       # 41 is not an item in the list.
(b) L1=[1,11,21,31]
    An=L1.remove(31)
    print(An+2)            # Unsupported Operand Type for oprator '+'
```

14. Find the errors:

(a)

```
L1=[3,4,5]
L2=L1*3
print(L1*3.0)
print(L2)
```

Ans:

```
>>> L1=[3,4,5]
>>> L2=L1*3
>>> print(L1*3.0)    # Error: non-int of type 'float'
>>> print(L2)
[3, 4, 5, 3, 4, 5, 3, 4, 5]
```

(b)

```
L1=[3,3,8,1,3,0, '1', '0' , '2', 'e', 'w', 'e', 'r']
print(L1[::-1])
print(L1[-1:-2:-3])
print(L1[-1:-2:-3:-4])
```

Ans:

```
    >>> L1=[3,3,8,1,3,0,'1','0','2','e','w','e','r']
    >>> print(L1[::-1])
        ['r', 'e', 'w', 'e', '2', '0', '1', 0, 3, 1, 8, 3, 3]
    >>> print(L1[-1:-2:-3])
        ['r']
    >>> print(L1[-1:-2:-3:-4])          #Error: SyntaxError: invalid syntax
```

15. What will be the output of the following code?

```
X=['3','2', '5']
Y= ''
while X:
    Y=Y+X[-1]
    X=X[:len(x)-1]
print(Y)
print(X)
print(type(X), type(Y))
```

Ans:

```
    523
    []
    <class 'list'> <class 'str'>
```

6.Dictionaries

#Dictionary – Key: Value pair
#Working with dictionaries
#Dictionary Function and Methods

Dictionary is also one of the methods to organize collections like List, String etc.
Dictionaries are mutable, unordered collection with elements in the form of a key: value pair that associate key to value.
The key is used to find the value in dictionary in the same way as index is used to find the value in the list.

Creating a Dictionary:

<dictionary name> = {<key1:value1>,<key2:value2>,<key3:value3>}

The key of the dictionary must be immutable type such as :

- A Python string
- A Number
- A tuple

# Accessing elements of a dictionary:

Element of a dictionary can be accessed by the following method:
<dictionary name>[<key>]
Attempting to access a key that does not exist causes an error.

Month ={“Jan”:31, “Feb”:28, “Mar”: 31, “April”: 30}

To access item of Dictionary Month we should use:
Month[Feb]
Will return 28
Traversing a dictionary
Using for loop traversing can be done:

```
for <item> in <Dicionary>:
        Process each item here
```

Example:

```
for key in Month:
        print(key, “:”, Month[key])
```

Accessing keys or Values simultaneously :
The keys() and values() function of the dictionary allows to access the keys and values of a dictionary in one go.

Example:

```
>>>Month.keys()
dict_keys[“Jan”, “Feb”, “Mar”, “Apr”]
>>>Month.values()
dict_keys[31,28,31,30]
```

We can convert the sequence returned by keys() and values() function by using list() as shown below:

```
>>>list(Month.keys())
[“Jan”, “Feb”, “Mar”, “Apr”]
>>>list(Month.values())
[31,28,31,30]
```

Characteristics of a Dictionary:

1. *Unordered set*
2. *Not a Sequence*
3. *Indexed by Keys, Not Numbers*
4. *Keys must be unique*
5. *Mutable*
6. *Internally stored as Mapping*

Multiple ways of creating dictionaries:

1. **Initializing a Dictionary**
   Data = {“d1”:1, “d2”: 2, “d3”: 3, “d4”: 4}
2. **Adding key:value pair to an empty dictionary**

To create an empty dictionary, we have two ways:

a. Data = {}
b. Data = dict()

To add key:value pair one at a time:
Data[<key>] = <value>

3. **Creating a dictionary from name and value pairs:**
    a. Specify Key:Value pair
       Example:
       Data = dict(“d5” = 5, “d6” = 6)
    b. Specify comma separated value pair
       Example:
       Data=dict({“d5”:5, “d6”:6})
    c. Specify keys separately and corresponding values separately
       Example:
       Data =dict(zip((“E”, “F”),(5,6)))

<u>Adding elements to a dictionary:</u>

```
>>>Data[“d7”] = 7
>>>Data
{“d1”:1, “d2”: 2, “d7”:7,  “d3”: 3, “d4”: 4}
```

Nesting a dictionary:
Empl = {“Amit”:{“age”:14, “Cls”: 9}, “Rohan”: {“age”: 15, “Cls”:10}}

<u>Updating existing elements in a Dictionary:</u>

We can change the value of existing key as per the following syntax:
Dictionary[<key>] = <value>

Example:

```
Data = {“d1”:1, “d2”: 2, “d3”: 3, “d4”: 4}
>>>Data[“d2”] = 6
>>>Data
{“d1”:1, “d2”: 6, “d3”: 3, “d4”: 4}
```

<u>Deleting elements from dictionary:</u>

Two methods are used to remove the elements from the dictionary. The syntax is:

(i) del <dictionary>[key]

Example:

```
>>>del Data[“d1”]
>>>Data
{“d2”: 6, “d3”: 3, “d4”: 4}
```

(ii) By using pop() method

```
dictionary.pop(<key>)
>>>Data
{“d1”:1, “d2”: 2, “d3”: 3, “d4”: 4}
```

```
>>>Data.pop(“d2”)
>>>Data
{“d1”:1, “d3”: 3, “d4”: 4}
```

Checking for existence of a key:
Using membership operator in and not in we can check the existence of a key in the dictionary. For example:

```
>>>Data
{“d1”:1, “d2”: 2, “d3”: 3, “d4”: 4}
>>>“d2” in Data
True
```

To check the existence of value in dictionary we have to use values() method.
Example:

```
>>> 4 in Data.values()
True
```

The JSON module of Python:

It is JavaScript Object Notion(JSON). It is used to encode Python objects as JSON strin, and decode JSON strings into Python objects.
The json module provides an API similar to pickle for converting in-memory Python objects to a serialized representation known as JavaScript Object Notation.
We need to import this module by using import statement.

```
>>>import json;
```

Use of json:

```
>>>print(json.dumps(Data, indent=2))
{
“d1”:1
“d2”:2
“d3”:3
“d4”:4
}
```

Dictionary Functions and methods:

1. The len() method
   To obtain the number of elements of a dictionary we use len() method in the following manner:
   ```
   >>>len(Data)
   4   # Total 4 elements
   ```
2. The clear() method
   To remove all items of a dictionary and make the dictionary empty we use clear() method.
   ```
   >>>Data.clear()
   >>>Data
   {}
   ```
3. The get() method

We can get the corresponding value of a key by using the get method. If the key is not present then it produces an error message. However user can also display the user defined method.

```
>>>Data.get("d4")
4
>>>Data.get("d8")
NameError: name 'd8' is not defined.        # Error message
```

We could write the following to display user defined error message:

```
>>>Data.get("d8" , 'Error!!!')
Error!!!
```

4. The items() method

   It returns all of the items in the dictionary as a sequence of (key, value) tuples.

   The syntax is :

   ```
   <dictionary>.items()
   List1 = Data.items()
   for a in List1:
       print(a)
   ```

   It will print:

   ```
   ("d1",1)
   ("d2",2)
   ("d3",3)
   ("d4",4)
   ```

5. The keys() method

   ```
   >>>Data.keys()
   ```

   To display all the keys of a dictionary.

6. The value method

   ```
   >>>Data.values()
   ```

   To display all the values of a dictionary.

7. The update() method

It will update the contents of a dictionary.

For example:

```
>>>Data
{"d1":1, "d2": 2, "d3": 3, "d4": 4}
>>>Data1 = {"d1":1, "d5": 2, "d3": 3, "d4": 4}
>>>Data.update(Data1)
>>>Data
{"d5":2, "d3":3, "d1":1, "d4":4}
```

Exercise (Questions and Answers)