6. What are multi-line statements?
7. What do you mean by suite in Python?
8. Briefly explain the Input and Output functions used in Python.
9. What is the purpose of import function?
10. What is the purpose of // operator?
11. What is the purpose of ** operator?
12. What is the purpose of is operator?
13. What is the purpose of not in operator?
14. Briefly explain the various types of operators in Python.
15. List out the operator precedence in Python.

□□□

Chapter 2

Data Types and Operations

The data stored in memory can be of many types. For example, a person's name is stored as alphabets, age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has the following standard data types.

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

2.1 NUMBERS

Number data types store numeric values. Number objects are created when you assign a value to them. For example `a = 1, b = 20`. You can also delete the reference to a number object by using the del statement. The syntax of the del statement is as follows.

```
del  variable1[,variable2[,variable3[....,variableN]]]
```

You can delete a single object or multiple objects by using the del statement. For example

```
del a
del a,b
```

Python supports four different numerical types

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Integers can be of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

Example Program

```
a,b,c,d=1,1.5,231456987,2+9j
print("a=",a)
print("b=",b)
print("c=",c)
print("d=",d)
```

Output

```
a= 1
b= 1.5
c= 231456987
d= (2+9j)
```

2.1.1 Mathematical Functions

Python provides various built-in Mathematical Functions to perform mathematical calculations. The following Table 2.1 provides various Mathematical Functions and its purpose. For using the below functions, all functions except abs(x), max(x1,x2,...xn), min(x1,x2,...xn), round(x[,n]) and pow(x,y) need to import the math module because these functions reside in the math module. The math module also defines two mathematical constants pi and e. Modules will be discussed in Chapter 5.

Table 2.1 Mathematical Functions

Function	Description
abs(x)	Returns the absolute value of x.
sqrt(x)	Finds the square root of x.
ceil(x)	Finds the smallest integer not less than x.
floor(x)	Finds the largest integer not greater than x.
pow(x,y)	Finds x raised to y.
exp(x)	Returns e^x ie exponential of x.

fabs(x)	Returns the absolute value of x.
log(x)	Finds the natural logarithm of x for x>0.
log10(x)	Finds the logarithm to the base 10 for x>0.
max(x1,x2,...xn)	Returns the largest of its arguments.
min(x1,x2,....xn)	Returns the smallest of its arguments.
round(x,[n])	In case of decimal numbers, x will be rounded to n digits.
modf(x)	Returns the integer and decimal part as a tuple. The integer part is returned as a decimal.

Example Program

```
import math
print("Absolute value of -120:",abs(-120))
print("Square root of 25:",math.sqrt(25))
print("Ceiling of 12.2:", math.ceil(12.2))
print("Floor of 12.2:", math.floor(12.2))
print("2 raised to 3:", pow(2,3))
print("Exponential of 3:",math.exp(3))
print("Absolute value of -123:", math.fabs(-123))
print("Natural Logarithm of 2:", math.log(2))
print("Logarithm to the Base 10 of 2:",math.log10(2))
print("Largest among 10, 4, 2:",max(10,4,2))
print("Smallest among 10,4, 2:",min(10,4,2))
print("12.6789 rounded to 2 decimal places:",round(12.6789,2))
print("Decimal part and integer part of 12.090876:", math.modf(12.090876))
```

Output

```
Absolute value of -120: 120
Square root of 25: 5.0
Ceiling of 12.2: 13
Floor of 12.2: 12
2 raised to 3: 8
Exponential of 3: 20.085536923187668
Absolute value of -123: 123.0
Natural Logarithm of 2: 0.6931471805599453
Logarithm to the Base 10 of 2: 0.3010299956639812
Largest among 10, 4, 2: 10
Smallest among 10,4, 2: 2
12.6789 rounded to 2 decimal places: 12.68
Decimal part and integer part of 12.090876: (0.09087599999999973, 12.0)
```

2.1.2 Trigonometric Functions

There are several built-in Trigonometric Functions in Python. The function names and its purpose are listed in Table 2.2.

Table 2.2 Trigonometric Functions

Function	Description
sin(x)	Returns the sine of x radians.
cos(x)	Returns the cosine of x radians.
tan(x)	Returns the tangent of x radians.
asin(x)	Returns the arc sine of x, in radians.
acos(x)	Returns the arc cosine of x, in radians.
atan(x)	Returns the arc tangent of x, in radians.
atan2(y,x)	Returns atan(y / x), in radians.
hypot(x,y)	Returns the Euclidean form, sqrt(x*x + y*y).
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

Example

```
import math
print("Sin(90):",math.sin(90))
print("Cos(90):",math.cos(90))
print("Tan(90):",math.tan(90))
print("asin(1):",math.asin(1))
print("acos(1):",math.acos(1))
print("atan(1):",math.atan(1))
print("atan2(3,2):",math.atan2(3,2))
print("Hypotenuse of 3 and 4:",math.hypot(3,4))
print("Degrees of 90:",math.degrees(90))
print("Radians of 90:",math.radians(90))
```

Output

```
Sin(90): 0.893996663601
Cos(90): -0.448073616129
Tan(90): -1.99520041221
asin(1): 1.57079632679
acos(1): 0.0
atan(1): 0.785398163397
atan2(3,2): 0.982793723247
Hypotenuse of 3 and 4: 5.0
Degrees of 90: 5156.62015618
Radians of 90: 1.57079632679
```


2.1.3 Random Number Functions

There are several random number functions supported by Python. Random numbers have applications in research, games, cryptography, simulation and other applications. Table 2.3 shows the commonly used Random Number Functions in Python. For using the Random Number Functions, we need to import the module random because all these functions reside in the random module.

Table 2.3 Random Number Functions

Function	Description
choice(sequence)	Returns a random value from a sequence like list, tuple, string etc.
shuffle(list)	Shuffles the items randomly in a list. Returns none.
random()	Returns a random floating-point number which lies between 0 and 1.
randrange([start,] stop [,step])	Returns a randomly selected number from a range where start shows the starting of the range, stop shows the end of the range and step decides the number to be added to decide a random number.
seed([x])	Gives the starting value for generating a random number. Returns none. This function is called before calling any other random module function.
uniform(x,y)	Generates a random floating-point number n such that n>x and n<y.

Example

```
import random
s='abcde'
print("choice(abcde):", random.choice(s))
list = [10,2,3,1,8,19]
print("shuffle(list):", random.shuffle(list))
print("random.seed(20):", random.seed(20))
print("random():",random.random())
print("uniform(2,3):", random.uniform(2,3))
print("randrange(2,10,1):",random.randrange(2,10,1))
```

Output

```
choice(abcde): b
shuffle(list): None
random.seed(20): None
random(): 0.905639676175
uniform(2,3): 2.68625415703
randrange(2,10,1): 8
```

2.2 STRINGS

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings

can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and ending at -1.The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. The % operator is used for string formatting which will be discussed in Section 2.2.2.

Example Program

```
str = 'Welcome to Python Programming '
print(str)                  # Prints complete string
print(str[0])               # Prints first character of the string
print(str[11:17])           # Prints characters starting from 11th to 17th
print(str[11:])             # Prints string starting from 11th character
print(str * 2)              # Prints string two times
print(str + "Session")# Prints concatenated string
```

Output

```
Welcome to Python Programming
W
Python
Python Programming
Welcome to Python Programming Welcome to Python Programming
Welcome to Python Programming Session
```

2.2.1 Escape Characters

An escape character is a character that gets interpreted when placed in single or double quotes. They are represented using backslash notation. These characters are non printable. The following Table 2.4 shows the most commonly used escape characters and their description.

Table 2.4 Escape Characters

Escape Characters	Description
\a	Bell or alert
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage Return
\s	Space
\t	Tab
\v	Vertical Tab

2.2.2 String Formatting Operator

This operator is unique to strings and is similar to the formatting operations of the printf() function of the C programming language. The following Table 2.5 shows various format symbols and their conversion.

Table 2.5 Format Symbols and their Conversion

Format Symbols	Conversion
%c	Character
%s	String
%i	Signed Decimal Integer
%d	Signed Decimal Integer
%u	Unsigned Decimal Integer
%o	Octal Integer
%x	Hexadecimal Integer (lowercase letters)
%X	Hexadecimal Integer(uppercase letters)
%e	Exponential notation with lowercase letter 'e'
%E	Exponential notation with uppercase letter 'E'
%f	Floating-point real number

The following example shows the usage of various string formatting functions.

Example Program

```
#Demo of String Formatting Operators
print("The first letter of %s is %c " %('apple', 'a')) #usage of %s and %c
print("The sum=%d" %(-12)) #usage of %d
print("The sum=%i" %(-12)) #usage of %i
print("The sum=%u" %(12)) #usage of %u
print("%o is the octal octal equivalent of %d" %(8,8)) #usage of %o
print("%x is the hexadecimal equivalent of %d" %(10,10))#usage of %x
print("%X is the hexadecimal equivalent of %d" %(10,10))#usage of %X
print("%e is the exponential equivalent of %f" %(10.98765432,10.98765432))
#usage of %e and %f
print("%E is the exponential equivalent of %f" %(10.98765432,10.98765432))
#usage of %E and %f
print( "%.2f" %(32.1274)) #usage of %f for printing upto two decimal
places
```

Output

```
The first letter of apple is a
The sum=-12
The sum=-12
The sum=12
10 is the octal octal equivalent of 8
a is the hexadecimal equivalent of 10
A is the hexadecimal equivalent of 10
```

```
1.098765e+01 is the exponential equivalent of 10.987654
1.098765E+01 is the exponential equivalent of 10.987654
32.13
```

2.2.3 String Formatting Functions

Python includes a large number of built-in functions to manipulate strings.

1. **len(string)**- Returns the length of the string

   Example Program

   ```
   #Demo of len(string)
   s='Learning Python is fun!'
   print("Length of", s, "is", len(s))
   ```

   Output

   ```
   Length of Learning Python is fun! is 23
   ```

2. **lower()** – Returns a copy of the string in which all uppercase alphabets in a string are converted to lowercase alphabets.

   Example Program

   ```
   #Demo of lower()
   s='Learning Python is fun!'
   print(s.lower())
   ```

   Output

   ```
   learning python is fun!
   ```

3. **upper()** – Returns a copy of the string in which all lowercase alphabets in a string are converted to uppercase alphabets.

   Example Program

   ```
   #Demo of upper()
   s='Learning Python is fun!'
   print(s.upper())
   ```

   Output

   ```
   LEARNING PYTHON IS FUN!
   ```

4. **swapcase()** – Returns a copy of the string in which the case of all the alphabets are swapped. ie. all the lowercase alphabets are converted to uppercase and vice versa.

   Example Program

   ```
   #Demo of swapcase()
   s='LEARNing PYTHON is fun!'
   print(s.swapcase())
   ```

   Output

   ```
   learnING Python IS FUN!
   ```

5. **capitalize()** – Returns a copy of the string with only its first character capitalized.

   Example Program

   ```
   #Demo of capitalize()
   s='learning Python is fun!'
   print(s.capitalize())
   ```


   Output

   ```
   Learning python is fun!
   ```

6. **title()** - Returns a copy of the string in which first characters of all the words are capitalized.

   Example Program

   ```
   #Demo of title()
   s='learning Python is fun!'
   print(s.title())
   ```

   Output

   ```
   Learning Python Is Fun!
   ```

7. **lstrip()** – Returns a copy of the string in which all the characters have been stripped(removed) from the beginning. The default character is whitespaces.

   Example Program

   ```
   #Demo of lstrip()
   s='      learning Python is fun!'
   print(s.lstrip())
   s='*******learning Python is fun!'
   print(s.lstrip('*'))
   ```

   Output

   ```
   learning Python is fun!
   learning Python is fun!
   ```

8. **rstrip()** - Returns a copy of the string in which all the characters have been stripped(removed) from the beginning. The default character is whitespaces.

   Example Program

   ```
   #Demo of rstrip()
   s='learning Python is fun!     '
   print(s.rstrip())
   s='learning Python is fun!******'
   print(s.rstrip('*'))
   ```

   Output

   ```
   learning Python is fun!
   learning Python is fun!
   ```

9. **strip()** - Returns a copy of the string in which all the characters have been stripped(removed) from the beginning and end . It performs both lstrip() and rstrip(). The default character is whitespaces.

   Example Program

   ```
   #Demo of strip()
   s='   learning Python is fun!     '
   print(s.strip())
   s='******learning Python is fun!******'
   print(s.strip('*'))
   ```

Output

```
learning Python is fun!
learning Python is fun!
```

10. **max(str)** – Returns the maximum alphabetical character from string str.

Example Program

```
#Demo of max(str)
s='learning Python is fun'
print('Maximum character is :', max(s))
```

Output

```
Maximum character is : y
```

11. **min(str)** - Returns the minimum alphabetical character from string str.

Example Program

```
#Demo of min(str)
s='learning-Python-is-fun'
print('Minimum character is :', min(s))
```

Output

```
Minimum character is : -
```

12. **replace(old, new [,max])** – Returns a copy of the string with all the occurrences of substring old is replaced by new. The max is optional and if it is specified, the first occurrences specified in max are replaced.

Example Program

```
# Demo of replace(old, new[,max])
s="This is very new.This is good"
print(s.replace('is', 'was'))
print(s.replace('is', 'was', 2))
```

Output

```
Thwas was very new.Thwas was good
Thwas was very new.This is good
```

13. **center(width, fillchar)** – Returns a string centered in a length specified in the width variable. Padding is done using the character specified in the fillchar. Default padding is space.

Example Program

```
# Demo of center(width, fillchar)
s="This is Python Programming"
print(s.center(30, '*'))
print(s.center(30))
```

Output

```
**This is Python Programming**
This is Python Programming
```

14. ***ljust(width[,fillchar])*** – Returns a string left-justified in a length specified in the *width variable. Padding is done using the character specified in the fillchar. Default padding is space.*

Example Program

```
# Demo of ljust(width[,fillchar])
s="This is Python Programming"
print(s.ljust(30, '*'))
print(s.ljust(30))
```

Output

```
This is Python Programming****
This is Python Programming
```

15. **rjust(width[,fillchar])** – Returns a string right-justified in a length specified in the width variable. Padding is done using the character specified in the fillchar. Default padding is space.

Example Program

```
# Demo of rjust(width[,fillchar])
s="This is Python Programming"
print(s.rjust(30, '*'))
print(s.rjust(30))
```

Output

```
****This is Python Programming
This is Python Programming
```

16. **zfill(width)** - The method zfill() pads string on the left with zeros to fill width.

Example Program

```
# Demo of zfill(width)
s="This is Python Programming"
print(s.zfill(30))
```

Output

```
0000This is Python Programming
```

17. **count(str,beg=0,end=len(string))** – Returns the number of occurrences of str in the range beg and end.

Example Program

```
# Demo of count(str,beg=0,end=len(string))
s="This is Python Programming"
print(s.count('i',0,10))
print(s.count('i',0,25))
```

Output

```
2
3
```

18. **find(str,beg=0,end=len(string))** – Returns the index of str if str occurs in the range beg and end and returns -1 if it is not found.

Example Program

```
# Demo of find(str,beg=0,end=len(string))
s="This is Python Programming"
```

```
print(s.find('thon',0,25))
print(s.find('thy'))
```

Output

```
10
-1
```

19. **rfind(str,beg=0,end=len(string))** – Same as find, but searches backward in a string.

Example Program

```
# Demo of rfind(str,beg=0,end=len(string))
s="This is Python Programming"
print(s.rfind('thon',0,25))
print(s.rfind('thy'))
```

Output

```
10
-1
```

20. **index(str,beg=0,end=len(string))** – Same as that of find but raises an exception if the item is not found.

Example Program

```
# Demo of index(str,beg=0,end=len(string))
s="This is Python Programming"
print(s.index('thon',0,25))
print(s.index('thy'))
```

Output

```
10
Traceback (most recent call last):
File "main.py", line 4, in <module>
print s.index('thy')
ValueError: substring not found
```

21. **rindex(str,beg=0,end=len(string))** – Same as index but searches backward in a string.

Example Program

```
# Demo of rindex(str,beg=0,end=len(string))
s="This is Python Programming"
print(s.rindex('thon',0,25))
print(s.rindex('thy'))
```

Output

```
10
Traceback (most recent call last):
File "main.py", line 4, in <module>
print s.index('thy')
ValueError: substring not found
```

22. ***startswith(suffix, beg=0,end=len(string))***- It returns True if the string begins with the specified suffix, otherwise return False.

Example Program

```
# Demo of startswith(suffix,beg=0,end=len(string))
s="Python programming is fun"
print(s.startswith('is',10,21))
s="Python programming is fun"
print(s.startswith('is',19,25))
```

Output

```
False
True
```

23. **endswith(suffix, beg=0,end=len(string))**- It returns `True` if the string ends with the specified suffix, otherwise return `False`.

Example Program

```
# Demo of endswith(suffix,beg=0,end=len(string))
s="Python programming is fun"
print(s.endswith('is',10,21))
s="Python programming is fun"
print(s.endswith('is',0,25))
```

Output

```
True
False
```

24. **isdecimal()** - Returns `True` if a unicode string contains only decimal characters and `False` otherwise. To define a string as unicode string, prefix 'u' to the front of the quotation marks.

Example Program

```
# Demo of isdecimal()
s=u"This is Python 1234"
print(s.isdecimal())
s=u"123456"
print(s.isdecimal())
```

Output

```
False
True
```

25. **isalpha()** – Returns `True` if string has at least 1 character and all characters are alphabetic and `False` otherwise.

Example Program

```
# Demo of isalpha()
s="This is Python1234"
print(s.isalpha())
s="Python"
print(s.isalpha())
```

Output

```
False
True
```

26. **isalnum()-** Returns `True` if string has at least 1 character and all characters are alphanumeric and `False` otherwise.

Example Program

```
# Demo of isalnum()
s="***Python1234"
print(s.isalnum())
s="Python1234"
print(s.isalnum())
```

Output

```
False
True
```

27. **isdigit()** – Returns `True` if string contains only digits and `False` otherwise

Example Program

```
# Demo of isdigit()
s="***Python1234"
print(s.isdigit())
s="123456"
print(s.isdigit())
```

Output

```
False
True
```

28. **islower()** – Returns `True` if string has at least 1 cased character and all cased characters are in lowercase and `False` otherwise.

Example Program

```
# Demo of islower()
s="Python Programming"
print(s.islower())
s="python programming"
print(s.islower())
```

Output

```
False
True
```

29. ***isupper()*** – Returns `True` if string has at least one cased character and all cased characters are in uppercase and `False` otherwise.

Example Program

```
# Demo of isupper()
s="Python Programming"
print(s.isupper())
```



```
s="PYTHON PROGRAMMING"
print(s.isupper())
```

Output

```
False
True
```

30. **isnumeric()** – Returns `True` if a unicode string contains only numeric characters and `False` otherwise.

Example Program

```
# Demo of isnumeric()
s=u"Python Programming1234"
print(s.isnumeric())
s=u"12345"
print(s.isnumeric())
```

Output

```
False
True
```

31. **isspace()** – Returns `True` if string contains only whitespace characters and `False` otherwise.

Example Program

```
# Demo of isspace()
s="Python Programming"
print(s.isspace())
s="   "
print(s.isspace())
```

Output

```
False
True
```

32. **istitle()** - Returns `True` if string is properly "titlecased" and `False` otherwise. Title case means each word in a sentence begins with uppercase letter.

Example Program

```
#Demo of istitle()
s="Python programming is fun"
print(s.istitle())
s="Python Programming Is Fun"
print(s.istitle())
```

Output

```
False
True
```

33. **expandtabs(tabsize=8)** - It returns a copy of the string in which tab characters ie. '\t' are expanded using spaces using the given tabsize. The default tabsize is 8.

Example Program

```
#Demo of expandtabs(tabsize)
s-"Python\tprogramming\tis\tfun"
print( s.expandtabs())
s="Python\tprogramming\tis\tfun"
print(s.expandtabs(10))
```

Output

```
Python  programming     is      fun
Python    programming           is          fun
```

34. **join(seq)** - Returns a string in which the string elements of sequence have been joined by a separator.

Example Program

```
# Demo of join(seq)
s="-"
seq=("Python", "Programming")
print(s.join(seq))
s="*"
seq=("Python", "Programming")
print(s.join(seq))
```

Output

```
Python-Programming
Python*Programming
```

35. **split(str="", num=string.count(str))** - Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified), optionally limiting the number of splits to num.

Example Program

```
#Demo of split(str="",num=string.count(str))
s="Python programming is fun"
print(s.split(' '))
s="Python*programming*is*fun"
print(s.split('*'))
s-"Python*programming*is*fun"
print(s.split('*',2))
```

Output

```
['Python', 'programming', 'is', 'fun']
['Python', 'programming', 'is', 'fun']
['Python', 'programming', 'is*fun']
```

36. **splitlines(num=string.count('\n'))** - Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed. If num is specified, it is assumed that line breaks need to be included in the lines.

Example Program

```
#Demo of splitlines(num=string.count('\n'))
s="Python\nprogramming\nis\nfun"
```



```
print(s.splitlines( ))
print(s.splitlines(0))
print(s.splitlines(1))
```

Output

```
['Python', 'programming', 'is', 'fun']
['Python', 'programming', 'is', 'fun']
['Python\n', 'programming\n', 'is\n', 'fun']
```

2.3 LIST

List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type. Items separated by commas are enclosed within brackets [ ]. To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and ending with -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Example Program

```
first_list = ['abcd', 147, 2.43, 'Tom', 74.9]
small_list = [111, 'Tom']
print(first_list)           # Prints complete list
print(first_list[0])        # Prints first element of the list
print(first_list[1:3])      # Prints elements starting from 2nd till 3rd
print(first_list[2:])       # Prints elements starting from 3rd element
print(small_list * 2)       # Prints list two times
print(first_list + small_list) # Prints concatenated lists
```

Output

```
['abcd', 147, 2.43, 'Tom', 74.9]
abcd
[147, 2.43]
[2.43, 'Tom', 74.9]
[111, 'Tom', 111, 'Tom']
[['abcd', 147, 2.43, 'Tom', 74.9, 111, 'Tom']
```

We can update lists by using the slice on the left hand side of the assignment operator. Updates can be done on single or multiple elements in a list.

Example Program

```
#Demo of List Update
list = ['abcd', 147, 2.43, 'Tom', 74.9]
print("Item at position 2=", list[2])
list[2]=500
print("Item at position 2=", list[2])
print("Item at Position 0 and 1 is", list[0],list[1])
list[0]=20; list[1]='apple'
```

```
print("Item at Position 0 and 1 is", list[0],list[1])
```

Output

```
Item at position 2= 2.43
Item at position 2= 500
Item at Position 0 and 1 is abcd 147
Item at Position 0 and 1 is 20 apple
```

To remove an item from a list, there are two methods. We can use del statement or remove() method which will be discussed in the subsequent session.

Example Program

```
#Demo of List Deletion
list = ['abcd', 147, 2.43, 'Tom', 74.9]
print(list)
del list[2]
print("List after deletion: ", list)
```

Output

```
['abcd', 147, 2.43, 'Tom', 74.9]
List after deletion:  ['abcd', 147, 'Tom', 74.9]
```

2.3.1 Built-in List Functions

1. **len(list)** – Gives the total length of the list.

   Example Program

   ```
   #Demo of len(list)
   list1 = ['abcd', 147, 2.43, 'Tom']
   print(len(list1))
   ```

   Output

   ```
   4
   ```

2. **max(list)** – Returns item from the list with maximum value.

   Example Program

   ```
   #Demo of max(list)
   list1 = [1200, 147, 2.43, 1.12]
   list2 = [213, 100, 289]
   print("Maximum value in: ", list1, "is",  max(list1))
   print("Maximum value in: ", list2, "is",  max(list2))
   ```

   Output

   ```
   Maximum value in:  [1200, 147, 2.43, 1.12] is 1200
   Maximum value in:  [213, 100, 289] is 289
   ```

3. **min(list)** – Returns item from the list with minimum value.

   Example Program

   ```
   #Demo of min(list)
   list1 = [1200, 147, 2.43, 1.12]
   list2 = [213, 100, 289]
   ```



```
print("Minimum value in: ", list1, "is",  min(list1))
print("Minimum value in: ", list2, "is",  min(list2))
```

Output

```
Minimum value in:  [1200, 147, 2.43, 1.12] is 1.12
Minimum value in:  [213, 100, 289] is 100
```

4. **list(seq)** – Returns a tuple into a list

   Example Program

   ```
   #Demo of list(seq)
   tuple = ('abcd', 147, 2.43, 'Tom')
   print("List:", list(tuple))
   ```

   Output

   ```
   List: ['abcd', 147, 2.43, 'Tom']
   ```

5. **map(aFunction,aSequence)** - One of the common things we do with list and other sequences is applying an operation to each item and collect the result. The map(aFunction, aSequence) function applies a passed-in function to each item in an iterable object and returns a list containing all the function call results.

   Example Program

   ```
   str=input("Enter a list(space separated):")
   lis=list(map(int,str.split()))
   print(lis)
   ```

   Output

   ```
   Enter a list(space separated):1 2 3 4
   [1, 2, 3, 4]
   ```

   In the above example, a string is read from the keyboard and each item is converted into int using map(aFunction,aSequence) function.

2.3.2 Built-in List Methods

1. **list.append(obj)** – This method appends an object obj passed to the existing list.

   Example Program

   ```
   #Demo of list.append(obj)
   list = ['abcd', 147, 2.43, 'Tom']
   print("Old List before Append:", list)
   list.append(100)
   print("New List after Append:", list)
   ```

   Output

   ```
   Old List before Append: ['abcd', 147, 2.43, 'Tom']
   New List after Append: ['abcd', 147, 2.43, 'Tom', 100]
   ```

2. **list.count(obj)** – Returns how many times the object obj appears in a list.

   Example Program

   ```
   #Demo of list.count(obj)
   list = ['abcd', 147, 2.43, 'Tom',147,200,147]
   ```

```
print("The number of times", 147, "appears in", list,
"=",list.count(147))
```

Output

```
The number of times 147 appears in['abcd', 147, 2.43, 'Tom', 147,
200, 147]= 3
```

3. **list.remove(obj)** – Removes object obj from the list.

Example Program

```
#Demo of list.remove(obj)
list1 = ['abcd', 147, 2.43, 'Tom']
list1.remove('Tom')
print(list1)
```

Output

```
['abcd', 147, 2.43]
```

4. **list.index(obj)** - Returns index of the object obj if found, otherwise raise an exception indicating that value does not exist.

Example Program

```
#Demo of list.index(obj)
list1 = ['abcd', 147, 2.43, 'Tom']
print(list1.index(2.43))
```

Output

```
2
```

5. **list.extend(seq)** – Appends the contents in a sequence seq passed to a list.

Example Program

```
#Demo of list.extend(seq)
list1 = ['abcd', 147, 2.43, 'Tom']
list2 = ['def', 100]
list1.extend(list2)
print(list1)
```

Output

```
['abcd', 147, 2.43, 'Tom', 'def', 100]
```

6. **list.reverse()** – Reverses objects in a list

Example Program

```
#Demo of list.reverse()
list1 = ['abcd', 147, 2.43, 'Tom']
list1.reverse()
print(list1)
```

Output

```
['Tom', 2.43, 147, 'abcd']
```

7. ***list.insert*(index,obj)** – Returns a list with object obj inserted at the given index.

Example Program

```
#Demo of list.insert(index,obj)
```



```
list1 = ['abcd', 147, 2.43, 'Tom']
print("List before insertion:",list1)
list1.insert(2,222)
print("List after insertion:",list1)
```

Output

```
List before insertion: ['abcd', 147, 2.43, 'Tom']
List after insertion: ['abcd', 147, 222, 2.43, 'Tom']
```

8. **list.sort([Key=None,Reverse=False])** – Sorts the items in a list and returns the list. If a function is provided, it will compare using the function provided.

Example Program

```
#Demo of list.sort([Key=None,Reverse=False])
list1 = [890, 147, 2.43,100]
print("List before sorting:",list1)
list1.sort()
print("List after sorting in ascending order:",list1)
```

Output

```
List before sorting: [840, 147, 2.43, 100]
List after sorting in ascending order: [2.43, 100, 147,840]
```

The following example illustrates how to sort the list in descending order.

Example Program

```
#Demo of list.sort([Key=None,Reverse=False])
list1 = [890, 147, 2.43,100]
print("List before sorting:",list1)
list1.sort(reverse=True)
print("List after sorting in descending order:",list1)
```

Output

```
List before sorting: [840, 147, 2.43, 100]
List after sorting in descending order: [840, 147, 100, 2.43]
```

9. **list.pop([index])** – Removes or Returns the last object obj from a list. We can even pop out any item in a list with the index specified.

Example Program

```
#Demo of list.pop([index])
list1 = ['abcd', 147, 2.43, 'Tom']
print("List before poping",list1)
list1.pop(-1)
print("List after poping:",list1)
item=list1.pop(-3)
print("Popped item:",item)
print("List after poping:",list1)
```

Output

```
List before poping ['abcd', 147, 2.43, 'Tom']
List after poping: ['abcd', 147, 2.43]
```

10. **list.clear()** – Removes all items from a list.

Example Program

```
#Demo of list.clear()
list1 = ['abcd', 147, 2.43, 'Tom"]
print("List before clearing:",list1)
list1.clear()
print("List after clearing:",list1)
```

Output

```
List before clearing:['abcd', 147, 2.43, 'Tom']
List after clearing:[]
```

11. **list.copy()** – Returns a copy of the list

Example Program

```
#Demo of list.copy()
list1 = ['abcd', 147, 2.43, 'Tom']
print("List before clearing:",list1)
list2=list1.copy()
list1.clear()
print("List after clearing:",list1)
print("Copy of the list:",list2)
```

Output

```
List before clearing: ['abcd', 147, 2.43, 'Tom']
List after clearing: []
Copy of the list: ['abcd', 147, 2.43, 'Tom']
```

2.3.3 Using List as Stacks

The list can be used as a stack (Last IN First Out). Stack is a data structure where the last element added is the first element retrieved. The list methods make it very easy to use a list as a stack. To add an item to the top of the stack, use **append()**. To retrieve an item from the top of the stack, use **pop()** without an explicit index.

Example Program

```
#Demo of List as Stack
stack=[10,20,30,40,50]
stack.append(60)
print("Stack after appending:",stack)
stack.pop()
print("Stack after poping:",stack)
```

Output

```
Stack after appending: [10, 20, 30, 40, 50, 60]
Stack after poping: [10, 20, 30, 40, 50]
```


2.3.4 Using List as Queues

It is also possible to use a list as a queue, where the first element added is the first element retrieved. Queues are Last In First Out (LIFO) data structure. But lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow since all of the other elements have to be shifted by one.

To implement a queue, Python provides a module called **collections** in which a method called **deque** is designed to have fast appends and pops from both ends.

Example Program

```
#Demo of List as Queue
from collections import deque
queue = deque(["apple", "orange", "pear"])
queue.append("cherry")              # cherry arrives
queue.append("grapes")              # grapes arrives
queue.popleft()                     # The first to arrive now leaves
queue.popleft()                     # The second to arrive now leaves
print(queue)                        # Remaining queue in order of arrival
```

Output

```
deque(['pear', 'cherry', 'grapes'])
```

More about list data structures like list comprehensions and nested lists will be explained in Chapter 3.

2.4 TUPLE

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. The main differences between lists and tuples are lists are enclosed in square brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be considered as read-only lists.

Example Program

```
first_tuple = ('abcd', 147, 2.43, 'Tom', 74.9)
small_tuple = (111, 'Tom')

print(first_tuple)          # Prints complete tuple
print(first_tuple[0])       # Prints first element of the tuple
print(first_tuple[1:3])     # Prints elements starting from 2nd till 3rd
print(first_tuple[2:])      # Prints elements starting from 3rd element
print(small_tuple * 2 )           # Prints tuple two times
print(first_tuple + small_tuple) # Prints concatenated tuples
```

Output

```
('abcd', 147, 2.43, 'Tom', 74.9) abcd
(147, 2.43)
(2.43, 'Tom', 74.9)
```

```
(111, 'Tom', 111, 'Tom')
('abcd', 147, 2.43, 'Tom', 74.9, 111, 'Tom')
```

The following code is invalid with tuple whereas this is possible with lists

```
first_list = [ 'abcd', 147 , 2.43, 'Tom', 74.9 ]
first_tuple = ('abcd', 147, 2.43, 'Tom', 74.9)
tuple[2] = 100     # Invalid syntax with tuple
list[2] = 100      # Valid syntax with list
```

To delete an entire tuple we can use the `del` statement. Example `del tuple`. It is not possible to remove individual items from a tuple. However it is possible to create tuples which contain mutable objects, such as lists.

Example Program

```
#Demo of Tuple containing Lists
t=([1,2,3],['apple','pear','orange'])
print(t)
```

Output

```
([1, 2, 3], ['apple', 'pear', 'orange'])
```

It is possible to pack values to a tuple and unpack values from a tuple. We can create tuples even without parenthesis. The reverse operation is called sequence unpacking and works for any sequence on the right-hand side. Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence.

Example Program

```
#Demo of Tuple packing and unpacking
t="apple",1,100
print(t)
x,y,z=t
print(x)
print(y)
print(z)
```

Output

```
('apple', 1, 100)
apple
1
100
```

2.4.1 Built-in Tuple Functions

1. **len(tuple)** – Gives the total length of the tuple.

   Example Program

   ```
   #Demo of len(tuple)
   tuple1 = ('abcd', 147, 2.43, 'Tom')
   print(len(tuple1))
   ```


   Output

   ```
   4
   ```

2. **max(tuple)** – Returns item from the tuple with maximum value.

   Example Program

   ```
   #Demo of max(tuple)
   tuple1 = (1200, 147, 2.43, 1.12)
   tuple2 = (213, 100, 289)
   print("Maximum value in: ", tuple1, "is",  max(tuple1))
   print("Maximum value in: ", tuple2, "is",  max(tuple2))
   ```

   Output

   ```
   Maximum value in:  (1200, 147, 2.43, 1.12) is 1200
   Maximum value in:  (213, 100, 289) is 289
   ```

3. **min(tuple)** – Returns item from the tuple with minimum value.

   Example Program

   ```
   #Demo of min(tuple)
   tuple1 = (1200, 147, 2.43, 1.12)
   tuple2 = (213, 100, 289)
   print("Minimum value in: ", tuple1, "is",  min(tuple1))
   print("Minimum value in: ", tuple2, "is",  min(tuple2))
   ```

   Output

   ```
   Minimum value in:  (1200, 147, 2.43, 1.12) is 1.12
   Minimum value in:  (213, 100, 289) is 100
   ```

4. **tuple(seq)** – Returns a list into a tuple.

   Example Program

   ```
   #Demo of tuple(seq)
   list = ['abcd', 147, 2.43, 'Tom']
   print("Tuple:", tuple(list))
   ```

   Output

   ```
   Tuple: ('abcd', 147, 2.43, 'Tom')
   ```

2.5 SET

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. It can have any number of items and they may be of different types (integer, float, tuple, string etc.). Items in a set are not ordered. Since they are unordered we cannot access or change an element of set using indexing or slicing. We can perform set operations like union, intersection, difference on two sets. Set have unique values. They eliminate duplicates. The slicing operator [] does not work with sets. An empty set is created by the function set().

Example Program

```
# Demo of Set Creation
s1={1,2,3} #set of integer numbers
print(s1)
```

```
s2={1,2,3,2,1,2}#output contains only unique values
print(s2)
s3={1, 2.4, 'apple', 'Tom', 3}#set of mixed data types
print(s3)
#s4={1,2,[3,4]}#sets cannot have mutable items
#print s4 # Hence not permitted
s5=set([1,2,3,4])#using set function to create set from a list
print(s5)
```

Output

```
{1, 2, 3}
{1, 2, 3}
{1, 3, 2.4, 'apple', 'Tom'}
{1, 2, 3, 4}
```

2.5.1 Built-in Set Functions

1. **len(set)** – Returns the length or total number of items in a set.

   Example Program

   ```
   #Demo of len(set)
   set1 = {'abcd', 147, 2.43, 'Tom'}
   print(len(set1))
   ```

   Output

   ```
   4
   ```

2. **max(set)** – Returns item from the set with maximum value.

   Example Program

   ```
   #Demo of max(set)
   set1 = {1200, 147, 2.43, 1.12}
   set2 = {213, 100, 289}
   print("Maximum value in: ", set1, "is",  max(set1))
   print("Maximum value in: ", set2, "is",  max(set2))
   ```

   Output

   ```
   Maximum value in:  {1200, 1.12, 2.43, 147} is 1200
   Maximum value in:  {289, 100, 213} is 289
   ```

3. **min(set)** – Returns item from the set with minimum value.

   Example Program

   ```
   #Demo of min(set)
   set1 = {1200, 147, 2.43,1.12}
   set2 = {213, 100, 289}
   print("Minimum value in: ", set1, "is",  min(set1))
   print("Minimum value in: ", set2, "is",  min(set2))
   ```

   Output

   ```
   Minimum value in:  {1200, 1.12, 2.43, 147} is 1.12
   Minimum value in:  {289, 100, 213} is 100
   ```


4. **sum(set)** – Returns the sum of all items in the set.

   Example Program

   ```
   #Demo of sum(set)
   set1 = {147, 2.43}
   set2 = {213, 100, 289}
   print("Sum of elements in", set1, "is", sum(set1))
   print("Sum of elements in", set2, "is",  sum(set2))
   ```

   Output

   ```
   Sum of elements in {147, 2.43} is 149.43
   Sum of elements in {289, 100, 213} is 602
   ```

5. **sorted(set)**- Returns a new sorted set. The set does not sort itself.

   Example Program

   ```
   #Demo of sorted(set)
   set1 = {213, 100, 289, 40, 23, 1, 1000}
   set2 = sorted(set1)
   print("Sum of elements before sorting:", set1)
   print("Sum of elements after sorting:", set2)
   ```

   Output

   ```
   Sum of elements before sorting:{1, 100, 289, 1000, 40, 213, 23}
   Sum of elements after sorting:{1, 23, 40, 100, 213, 289, 1000}
   ```

6. **enumerate(set)** – Returns an enumerate object. It contains the index and value of all the items of set as a pair.

   Example Program

   ```
   #Demo of enumerate(set)
   set1 = {213, 100, 289, 40, 23, 1, 1000}
   print("enumerate(set):",  enumerate(set1))
   ```

   Output

   ```
   enumerate(set): <enumerate object at 0x7f0a73573690>
   ```

7. **any(set)** – Returns `True`, if the set contains at least one item, `False` otherwise.

   Example Program

   ```
   #Demo of any(set)
   set1 = set()
   set2={1,2,3,4}
   print("any(set):",  any(set1))
   print("any(set):",  any(set2))
   ```

   Output

   ```
   any(set): False
   any(set): True
   ```

8. **all(set)** – Returns `True`, if all the elements are true or the set is empty.

   Example Program

   ```
   #Demo of all(set)
   ```

```
set1 = set()
set2={1,2,3,4}
print("all(set):", all(set1))
print("all(set):", all(set1))
```

Output

```
all(set): True
all(set): True
```

2.5.2 Built-in Set Methods

1. **set.add(obj)** – Adds an element obj to a set.

   Example Program

   ```
   #Demo of set.add(obj)
   set1={3,8,2,6}
   print("Set before addition:",set1)
   set1.add(9)
   print("Set after addition:", set1)
   ```

   Output

   ```
   Set before addition:{8, 2, 3, 6}
   Set after addition:{8, 9, 2, 3, 6}
   ```

2. **set.remove(obj)** – Removes an element obj from the set. Raises KeyError if the set is empty.

   Example Program

   ```
   #Demo of set.remove(obj)
   set1={3,8,2,6}
   print("Set before deletion:",set1)
   set1.remove(8)
   print("Set after deletion:", set1)
   ```

   Output

   ```
   Set before deletion:{8, 2, 3, 6}
   Set after deletion:{2, 3, 6}
   ```

3. **set.discard(obj)** - Removes an element obj from the set. Nothing happens if the element to be deleted is not in the set.

   Example Program

   ```
   #Demo of set.discard(obj)
   set1={3,8,2,6}
   print("Set before discard:",set1)
   set1.discard(8)
   print("Set after discard:", set1)# Element is present
   set1.discard(9)
   print("Set after discard:",set1)#Element is not present
   ```


   Output

   ```
   Set before discard:{8, 2, 3, 6}
   Set after discard:{2, 3, 6}
   Set after discard:{2, 3, 6}
   ```

4. **set.pop()** – Removes and returns an arbitrary set element. Raises KeyError if the set is empty.

   Example Program

   ```
   #Demo of set.pop()
   set1={3,8,2,6}
   print("Set before pop:",set1)
   set1.pop()
   print("Set after poping:", set1)
   ```

   Output

   ```
   Set before pop:{8, 2, 3, 6}
   Set after poping:{2, 3, 6}
   ```

5. **set1.union(set2)** – Returns the union of two sets as a new set.

   Example Program

   ```
   #Demo of set1.union(set2)
   set1={3,8,2,6}
   print("Set1:",set1)
   set2={4,2,1,9}
   print("Set2:",set2)
   set3=set1.union(set2)#Unique values will be taken
   print("Union:", set3)
   ```

   Output

   ```
   Set1:{8, 2, 3, 6}
   Set2:{9, 2, 4, 1}
   Union:{1, 2, 3, 4, 6, 8, 9}
   ```

6. **set1.update(set2)** – Update a set with the union of itself and others. The result will be stored in set1.

   Example Program

   ```
   #Demo of set1.update(set2)
   set1={3,8,2,6}
   print("Set1:",set1)
   set2={4,2,1,9}
   print("Set2:",set2)
   set1.update(set2)#Unique values will be taken
   print("Update Method:", set1)
   ```

   Output

   ```
   Set1:{8, 2, 3, 6}
   Set2:{9, 2, 4, 1}
   Update Method:{1, 2, 3, 4, 6, 8, 9}
   ```

7. **set1.intersection(set2)** – Returns the intersection of two sets as a new set.

Example Program

```
#Demo of set1.intersection(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set3 = set1.intersection(set2)
print("Intersection:", set3)
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 2, 4, 1}
Intersection:{2}
```

8. **set1.intersection_update()** – Update the set with the intersection of itself and another. The result will be stored in set1.

Example Program

```
#Demo of set1.intersection_update(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set1.intersection_update(set2)
print("Intersection:", set1)
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 2, 4, 1}
Intersection_update:{8, 2, 3, 6}
```

9. **set1.difference(set2)** – Returns the difference of two or more sets into a new set.

Example Program

```
#Demo of set1.difference(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set3= set1.difference(set2)
print("Difference:", set3)
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 2, 4, 1}
Difference:{8, 3, 6}
```


10. **set1.difference_update(set2)** – Remove all elements of another set set2 from set1 and the result is stored in set1.

Example Program

```
#Demo of set1.difference_update(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set1.difference_update(set2)
print("Difference Update:", set1)
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 2, 4, 1}
Difference Update:{8, 3, 6}
```

11. **set1.symmetric_difference(set2)-** Return the symmetric difference of two sets as a new set.

Example Program

```
#Demo of set1.Symmetric_difference(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set3=set1.symmetric_difference(set2)
print("Symmetric Difference :", set3)
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 2, 4, 1}
Symmetric Difference:{1, 3, 4, 6, 8, 9}
```

12. **set1.difference_update(set2)** – Update a set with the symmetric difference of itself and another.

Example Program

```
#Demo of set1.symmetric_difference_update(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,2,1,9}
print("Set2:",set2)
set1.symmetric_difference_update(set2)
print("Symmetric Difference Update:", set1)
```

Output

```
Set1: {8, 2, 3, 6}
Set2: {9, 2, 4, 1}
Symmetric Difference Update:{1, 3, 4, 6, 8, 9}
```

13. **set1.isdisjoint(set2)** – Returns True if two sets have a null intersection.

Example Program

```
#Demo of set1.isdisjoint(set2)
set1={3,8,2,6}
print("Set1:",set1)
set2={4,7,1,9}
print("Set2:",set2)
print("Result of set1.isdisjoint(set2):", set1.isdisjoint(set2))
```

Output

```
Set1:{8, 2, 3, 6}
Set2:{9, 7, 4, 1}
Result of set1.isdisjoint(set2): True
```

14. **set1.issubset(set2)** - Returns True if set1 is a subset of set2.

Example Program

```
#Demo of set1.issubset(set2)
set1={3,8}
print("Set1:",set1)
set2={3,8,4,7,1,9}
print("Set2:",set2)
print("Result of set1.issubset(set2):", set1.issubset(set2))
```

Output

```
Set1:{8, 3}
Set2:{1, 3, 4, 7, 8, 9}
Result of set1.issubset(set2): True
```

15. **set1.issuperset(set2)** – Returns True, if set1 is a super set of set2.

Example Program

```
#Demo of set1.issuperset(set2)
set1={3,8,4,6}
print("Set1:",set1)
set2={3,8,}
print("Set2:",set2)
print("Result of set1.issuperset(set2):", set1.issuperset(set2))
```

Output

```
Set1:{8, 3, 4, 6}
Set2:{8, 3}
Result of set1.issuperset(set2): True
```

2.5.3 Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets. Sets being mutable are unhashable, so they can't be used as dictionary keys which will be discussed in the next section. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the function frozenset(). This datatype supports methods like difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union(). Being immutable it does not have methods like add(), remove(), update(), difference_update(),intersection_update(),symmetric_difference_update() etc.

Example Program

```
#Demo of frozenset()
set1= frozenset({3,8,4,6})
print("Set1:",set1)
set2=frozenset({3,8})
print("Set2:",set2)
print("Result of set1.intersection(set2):", set1.intersection(set2))
```

Output

```
Set1: frozenset({8, 3, 4, 6})
Set2: frozenset({8, 3})
Result of set1.intersection(set2): frozenset({8, 3})
```

2.6 DICTIONARY

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type. Keys are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays".

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Example Program

```
dict={}
dict['one']="This is one"
dict[2]="This is two"
tinydict={'name': 'john','code':6734, 'dept': 'sales'}
studentdict={'name': 'john','marks':[35,80,90]}
print(dict['one'])        # Prints value for 'one' key
print(dict[2])            # Prints value for 2 key
print(tinydict)           # Prints complete dictionary
print(tinydict.keys())    # Prints all the keys
print(tinydict.values()) # Prints all the values
print(studentdict)        #Prints studentdict
```

Output

```
This is one
This is two
```

```
{'dept': 'sales', 'name': 'john', 'code': 6734}
dict_keys(['dept', 'name', 'code'])
dict_values(['sales', 'john', 6734])
{'marks': [35, 80, 90], 'name': 'john'}
```

We can update a dictionary by adding a new key-value pair or modifying an existing entry.

Example Program

```
#Demo of updating and adding new values to dictionary
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
dict1['age']=25 #updating existing value in Key-Value pair
print("Dictionary after update:",dict1)
dict1['Weight']=60 #Adding new Key-value pair
print("Dictionary after adding new Key-value pair:",dict1)
```

Output

```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Dictionary after update: {'age': 25, 'Age': 20, 'Name': 'Tom', 'Height': 160}
Dictionary after adding new Key-value pair: {'age': 25, 'Age': 20,
'Name': 'Tom', 'Weight': 60, 'Height': 160}
```

We can delete the entire dictionary elements or individual elements in a dictionary. We can use del statement to delete the dictionary completely. To remove entire elements of a dictionary, we can use the clear() method which will be discussed in the built-in methods of dictionary.

Example Program

```
#Demo of Deleting Dictionary
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
del dict1['Age'] #deleting Key value pair 'Age':20
print("Dictionary after deletion:",dict1)
dict1.clear() #Clearing entire dictionary
print(dict1)
```

Output

```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Dictionary after deletion: {'Name': 'Tom', 'Height': 160}
{}
```

Properties of Dictionary Keys

1. More than one entry per key is not allowed.ie, no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment is taken.
2. Keys are immutable. This means keys can be numbers, strings or tuple. But it does not permit mutable objects like lists.

2.6.1 Built-in Dictionary Functions

1. **len(dict)** – Gives the length of the dictionary.

   Example Program

   ```
   #Demo of len(dict)
   dict1= {'Name':'Tom','Age':20,'Height':160}
   print(dict1)
   print("Length of Dictionary=",len(dict1))
   ```

   Output

   ```
   {'Age': 20, 'Name': 'Tom', 'Height': 160}
   Length of Dictionary= 3
   ```

2. **str(dict)** - Produces a printable string representation of the dictionary.

   Example Program

   ```
   #Demo of str(dict)
   dict1= {'Name':'Tom','Age':20,'Height':160}
   print(dict1)
   print("Representation of Dictionary=",str(dict1))
   ```

   Output

   ```
   {'Age': 20, 'Name': 'Tom', 'Height': 160}
   Representation of Dictionary= {'Age': 20, 'Name': 'Tom', 'Height':
   160}
   ```

3. **type(variable)** - The method type() returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type. This function can be applied to any variable type like number, string, list, tuple etc.

   Example Program

   ```
   #Demo of type(variable)
   dict1= {'Name':'Tom','Age':20,'Height':160}
   print(dict1)
   print("Type(variable)=",type(dict1))
   s="abcde"
   print("Type(variable)=",type(s))
   list1= [1,'a',23,'Tom']
   print("Type(variable)=",type(list1))
   ```

   Output

   ```
   {'Age': 20, 'Name': 'Tom', 'Height': 160}
   Type(variable)= <type 'dict'>
   Type(variable)= <type 'str'>
   Type(variable)= <type 'list'>
   ```

2.6.2 Built-in Dictionary Methods

1. **dict.clear()** – Removes all elements of dictionary dict.
   Example Program
```
#Demo of dict.clear()
dict1= {'Name':'Tom','Age':20,'Height':160}
print dict1
dict1.clear()
print dict1
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
{}
```
2. **dict.copy()** – Returns a copy of the dictionary dict().
   Example Program
```
#Demo of dict.copy()
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
dict2=dict1.copy()
print(dict2)
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
{'Age': 20, 'Name': 'Tom', 'Height': 160}
```
3. **dict.keys()** – Returns a list of keys in dictionary dict.
   Example Program
```
#Demo of dict.keys()
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Keys in Dictionary:",dict1.keys())
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Keys in Dictionary: ['Age', 'Name', 'Height']
```
   The values of the keys will be displayed in a random order. In order to retrieve the keys in sorted order, we can use the **sorted()** function. But for using the **sorted()** function, all the keys should of the same type.
   Example Program
```
#Demo of dict.keys()in sorted order
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Keys in sorted order:",sorted(dict1.keys()))
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
```


4. **dict.values()** – This method returns list of all values available in a dictionary.
   Example Program
```
#Demo of dict.values()
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Values in Dictionary:",dict1.values())
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Values in Dictionary: [20, 'Tom', 160]
```
   The values will be displayed in a random order. In order to retrieve the values in sorted order, we can use the **sorted()** function. But for using the **sorted()** function, all the values should of the same type.
   Example Program
```
#Demo of dict.values()in sorted order
dict1= {'Height':160,'Age':20,'Weight':60}
print(dict1)
print("Values in sorted order:",sorted(dict1.values()))
```
   Output
```
{'Age': 20, 'Height': 160, 'Weight': 60}
Values in sorted order: [20, 60, 160]
```
5. **dict.items()** – Returns a list of dictionary dict's(key,value) tuple pairs.
   Example Program
```
#Demo of dict.items()
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Items in Dictionary:",dict1.items())
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Items in Dictionary: [('Age', 20), ('Name', 'Tom'), ('Height', 160)]
```
6. **dict1.update(dict2)** – The dictionary dict2's key-value pair will be updated in dictionary dict1.
   Example Program
```
#Demo of dict1.update(dict2)
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
dict2={'Weight': 60}
print(dict2)
dict1.update(dict2)
print("Dict1 updated Dict2 :",dict1)
```
   Output
```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
```

```
{'Weight': 60}
Dict1 updated Dict2 : {'Age': 20, 'Name': 'Tom', 'Weight': 60,
'Height': 160}
```

7. **dict.has_key(key) -** Returns True, if the key is in the dictionary dict, else False is returned.

   Example Program

```
##Demo of dict1.has_key(key)
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Dict1.has_key(key)  :",dict1.has_key('Age'))
print("Dict1.has_key(key)  :",dict1.has_key('Phone'))
```

   Output

```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Dict1.has_key(key)  : True
Dict1.has_key(key)  : False
```

8. **dict.get(key, default=None) –** Returns the value corresponding to the key specified and if the key specified is not in the dictionary, it returns the default value.

   Example Program

```
#Demo of dict1.get(key,default='Name')
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Dict1.get('Age')  :",dict1.get('Age'))
print("Dict1.get('Phone') :",dict1.get('Phone',0))#'Phone not a key'
hence 0 #is given as default
```

   Output

```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Dict1.get(key)  : 20
Dict1.get(key)  : 0
```

9. **dict.setdefault(key,default=None) –** Similar to dict.get(key,default=None) but will set the key with the value passed and if key is not in the dictionary, it will set with the default value.

   Example Program

```
#Demo of dict1.set(key,default='Name')
dict1= {'Name':'Tom','Age':20,'Height':160}
print(dict1)
print("Dict1.setdefault('Age')  :",dict1.setdefault('Age'))
print("Dict1.setdefault('Phone',0))#'Phone not a #key, hence 0 is given
as default value.
```

   Output

```
{'Age': 20, 'Name': 'Tom', 'Height': 160}
Dict1.setdefault('Age')  : 20
Dict1.setdefault('Phone')  : 0
```


10. **dict.fromkeys(seq,[val]) –** Creates a new dictionary from sequence seq and values from val.

    Example Program

```
#Demo of dict.fromkeys(seq,[val])
list= ['Name','Age','Height']
dict= dict.fromkeys(list)
print("New Dictionary:",dict)
```

    Output

```
New Dictionary: {'Age': None, 'Name': None, 'Height': None}
```

2.7 MUTABLE AND IMMUTABLE OBJECTS

Objects whose value can change are said to be mutable and objects whose value is unchangeable once they are created are called immutable. An object's mutability is determined by its type. For instance, numbers, strings and tuples are immutable, while lists, sets and dictionaries are mutable. The following example illustrates the mutability of objects.

Example Program

```
#Mutability illustration
#Numbers
a=10
print(a)#Value of a before subtraction
print(a-2)#value of a-2
print(a)#Value of a after subtraction
#Strings
str="abcde"
print(str)#Before applying upper()function
print(str.upper())#result of upper()
print(str)#After applying upper()function
#Lists
list=[1,2,3,4,5]
print(list)#Before applying remove()method
list.remove(3)
print(list)#After applying remove() method
```

Output

```
10
8
10
abcde
ABCDE
abcde
[1, 2, 3, 4, 5]
[1, 2, 4, 5]
```

From the above example, it is clear that the values of numbers and strings are not changed even after applying a function or operation. But in the case of list, the value is changed or the changes are reflected in the original variable and hence called mutable.

2.8 DATA TYPE CONVERSION

We may need to perform conversions between the built-in types. To convert between different data types, use the type name as a function. There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value. Table 2.6 shows various functions and their descriptions used for data type conversion.

Table 2.6 Functions for Data Type Conversions

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Example Program 1

```
# Demo of Data Type Conversions
x= 12.8
print("Integer x=",int(x))
x=12000
print("Long x=", long(x))
x=12
print("Floating Point x=", float(x))
real,img=5,2
print("Complex number =",complex(real,img))
x=12
print("String conversion of", x ,"is", str(x))
x=12
print("Expression String of", x ,"is", repr(x))
```

Output 1

```
Integer x= 12
Long x= 12000
Floating Point x= 12.0
Complex number = (5+2j)
String conversion of 12 is 12
Expression String of 12 is 12
```

Example Program 2

```
# Demo of Data Type Conversions
s='abcde'
s1={1:'a',2:'b',3:'c'}
print("Conversion of", s ,"to tuple is ",tuple(s))
print("Conversion of", s ,"to list is ",list(s))
print("Conversion of", s ,"to set is ",set(s))
print("Conversion of", s ,"to frozenset is ",frozenset(s))
print("Conversion of", s1 ,"to dictionary is ",dict(s1))
```

Output 1

```
Conversion of abcde to tuple is ('a', 'b', 'c', d', 'e')
Conversion of abcde to list is ['a', 'b', 'c','d', 'e']
Conversion of abcde to set is  set(['a', 'c', 'b', 'e', 'd'])
Conversion of abcde to frozenset is frozenset(['a', 'c', 'b', 'e',
'd'])
Conversion of {1: 'a', 2: 'b', 3: 'c'} to dictionary is {1: 'a', 2: 'b',
3: 'c'}
```

Example Program 3

```
# Demo of Data Type Conversions
a=120
s='a'
print("Conversion of", a ,"to character is",chr(a) )
print("Conversion of", a ,"to unicode character is",unichr(a))
print("Conversion of", s ,"to integer is",ord(s))
print("Conversion of", a ,"to hexadecimal string is",hex(a))
print("Conversion of", a ,"to octal string is",oct(a))
```

Output 1

```
Conversion of 120 to character is x
Conversion of 120 to unicode character is x
Conversion of a to integer is 97
Conversion of 120 to hexadecimal string is 0x78
Conversion of 120 to octal string is 0170
```

2.9 SOLVED LAB EXERCISES

1. Write a Python program which accept the radius of a circle from the user and compute the area.

Program

```
import math
r = float(input("Input the radius of the circle:"))
print("The area of the circle with radius ",r,"is:",(math.pi *
r**2))
```

Output

```
Input the radius of the circle:3.5
The area of the circle with radius  3.5 is: 38.48451000647496
```

2. Program to find the biggest of three numbers

Program

```
# Biggest of three numbers
a =int(input("Enter first number:")) #Reads First Number
b =int(input("Enter second number:")) #Reads Second Number
c =int(input("Enter third number:")) #Reads Third Number
big = max(a,b,c)#Finds the biggest
print("Biggest of", a,b,c, "is", big) #prints the biggest number
```

Output

```
Enter first number:3
Enter second number:4
Enter third number:5
Biggest of 3 4 5 is 5
```

3. Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.

Program

```
values = input("Input some comma seprated numbers : ")
list = values.split(",")
tuple = tuple(list)
print('List : ',list)
print('Tuple : ',tuple)
```

Output

```
Input some comma seprated numbers : 1,8,3,4,5
```



```
List :  ['1', '8', '3', '4', '5']
Tuple :  ('1', '8', '3', '4', '5')
```

4. Write a Python program to accept a filename from the user print the extension of that.

Program

```
filename = input("Input the Filename: ")
f_extns = filename.split(".")
print("The extension of the file is : ",f_extns[-1])
```

Output

```
Input the Filename: ABC.doc
The extension of the file is :  doc
```

5. Write a Python program to display the first and last colors from the following list.

Program

```
color=input("Enter colors(space between colors):")
color_list = color.split(" ")
print(color_list)
print("The first color in the list is:",color_list[0],"and last
color is:",color_list[-1])
```

Output

```
Enter colors(space between colors):red green blue white yellow
['red', 'green', 'blue', 'white', 'yellow']
The first color in the list is: red and last color is: yellow
```

6. Write a Python program that accept an integer (n) and computes the value of n+nn+nnn.

Program

```
a = int(input("Input an integer : "))
n1 = int("%s" % a)
n2 = int("%s%s" % (a,a))
n3 = int("%s%s%s" % (a,a,a))
print(n1," ", n2, " ",n3)
print("sum=",(n1+n2+n3))
```

Output

```
Input an integer : 2
2   22   222
sum= 246
```

7. Write a Python program to print out a set containing all the colors from color_list_1 which are not present in color_list_2.

Program

```
color1=input("Enter a set of colors(space separated):")
color2=input("Enter a set of colors(space separated):")
c1=set(color1.split())
```

```
c2=set(color2.split())
print("The difference between",c1, "and", c2,"is",c1.difference(c2))
```

Output

```
Enter a set of colors(space separated):red green blue
Enter a set of colors(space separated):green blue
The difference between {'blue', 'red', 'green'} and {'blue', 'green'}
is {'red'}
```

8. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

Program

```
str1=input("Enter a String:")
print("Original String:",str1)
char = str1[0]
str1 = str1.replace(char, '$')
str1 = char + str1[1:]
print("Replaced String:",str1)
```

Output

```
Enter a String:onion
Original String: onion
Replaced String: oni$n
```

9. Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

Program

```
str1=input("Enter first string:")
str2=input("Enter second string:")
new_a = str2[:2] + str1[2:]
new_b = str1[:2] + str2[2:]
print("The new string after swapping first two characters of both
string:",(new_a+' '+new_b))
```

Output

```
Enter first string:python
Enter second string:programming
The new string after swapping first two characters of both string
prthon pyogramming
```

10. Write a Python program to remove the nth index character from a non empty string.

Program

```
str1=input("Enter a string:")
n=int(input("Enter the index position of the character to be removed:"))
first_part = str1[:n-1]
last_part = str1[n:]
print("The new string after removing the character=",(first_part +
last_part))
```


Output

```
Enter a string:python
Enter the index position of the character to be removed:3
The new string after removing the character= pyhon
```

11. Write a Python program to change a given string to a new string where the first and last chars have been exchanged.

Program

```
str1=input("Enter a String:")
print("String after Swapping first and last character:",(str1[-1:]
+ str1[1:-1] + str1[:1]))
```

Output

```
Enter a String:python
String after Swapping first and last character: nythop
```

12. Write a Python script that takes input from the user and displays that input back in upper and lower cases.

Program

```
str1 = input("Input a String: ")
print("String in uppercase:",str1.upper())
print("String in lowercase is:",str1.lower())
```

Output

```
Input a String: Python programming is fun
String in uppercase: PYTHON PROGRAMMING IS FUN
String in lowercase is: python programming is fun
```

13. Write a Python program to get the largest number from a list.

Program

```
lis=input("Enter a list(space separated):")
lis1=list(map(int,lis.split()))
print("Maximum element in a list:",max(lis1))
```

Output

```
Enter a list(space separated):1 4 65 43 23 12
Maximum element in a list: 65
```

14. Write a Python program to clone or copy a list.

Program

```
l=input("Enter a list(space separated):")
lis =list(l.split())
new_lis=list(lis)
print("Old list:",lis)
print("Copy of list:",new_lis)
```

Output

```
Enter a list(space separated):1 2 Bob Cat
Old list: ['1', '2', 'Bob', 'Cat']
Copy of list: ['1', '2', 'Bob', 'Cat']
```

15. Write a Python program to shuffle and print a specified list.

Program

```
import random
color= input("Enter a list(space separated):")
liscolor =list(color.split())
random.shuffle(liscolor)
print("List after shuffling:",end='')
print(liscolor)
```

Output

```
Enter a list(space separated):1 cat bat 3 sat 5
List after shuffling:['3', '5', '1', 'bat', 'sat', 'cat']
```

16. Write a Python script to sort (ascending and descending) a dictionary by value.

Program

```
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
sorted_d = sorted(d.items())
print('Dictionary in ascending order by value : ',sorted_d)
sorted_d = sorted(d.items(),reverse=True)
print('Dictionary in descending order by value : ',sorted_d)
```

Output

```
Original dictionary :  {0: 0, 1: 2, 2: 1, 3: 4, 4: 3}
Dictionary in ascending order by value :  [(0, 0), (2, 1), (1, 2),
(4, 3), (3, 4)]
Dictionary in descending order by value :  [(4, 3), (3, 4), (1, 2),
(2, 1), (0, 0)]
```

17. Write a Python script to add key to a dictionary.

Program

```
d = {0:10, 1:20}
print(d)
d.update({2:30})
print(d)
```

Output

```
{0: 10, 1: 20}
{0: 10, 1: 20, 2: 30}
```

18. Write a Python script to merge two Python dictionaries.

Program

```
d1 = {'a': 100, 'b': 200}
d2 = {'x': 300, 'y': 200}
print("Dictionary 1=:",d1)
print("Dictionary 2=:",d2)
d = d1.copy()
```



```
d.update(d2)
print("Merged Dictionary:",d)
```

Output

```
Dictionary 1=: {'b': 200, 'a': 100}
Dictionary 2=: {'y': 200, 'x': 300}
Merged Dictionary: {'b': 200, 'x': 300, 'y': 200, 'a': 100}
```

2.10 CONCLUSION

This Chapter covers all the data types in Python which includes number, strings, list, tuple, set and dictionary. The data type number includes mathematical functions, trigonometric functions and random number functions. String includes escape characters, formatting operators and formatting functions. The built-in functions and methods associated with list, tuple, set and dictionary is illustrated with examples. Usage of list as stack and queues are explained. Finally the mutability properties of objects and data type conversions are illustrated with examples. The review questions given in Section 2.10 are the most frequently asked questions for Interviews and University Examinations. This help students to prepare for University Examinations, Certification Examinations and Job Interviews.

2.11 REVIEW QUESTIONS

1. What are the data types available in Python?
2. What is the output of print str if str = 'Python Programming'?
3. What is the output of print str[0] if str = 'Python Programming'?
4. What is the output of print str[2:5] if str = 'Python Programming'?
5. What is the output of print str[2:] if str = 'Python Programming'?
6. What is the output of print str * 2 if str = 'Python Programming'?
7. What is the output of print str + "TEST" if str = 'Python Programming'?
8. What is the output of print list if list = ['abcd', 786 , 2.23, 'Tom', 70.2]?
9. What is the output of print list[0] if list = ['abcd', 786 , 2.23, 'Tom', 70.2]?
10. What is the output of print list[1:3] if list = ['abcd', 786 , 2.23, 'Tom', 70.2]?
11. What is the output of print list[2:] if list = ['abcd', 786 , 2.23, 'Tom', 70.2]?
12. What is the output of print smalllist * 2 if smalllist = ['Jack', 20]?
13. What is the output of print list + smalllist * 2 if list = ['abcd', 786 , 2.23, 'Tom', 70.2] and smalllist = ['Jack', 20]?
14. What are tuples in Python?
15. What are sets in Python?
16. What is the difference between tuples and lists in Python?
17. What is the output of print tuple if tuple = ('abcd', 786 , 2.23, 'Tom', 70.2)?
18. What is the output of print tuple[0] if tuple = ('abcd', 786 , 2.23, 'Tom', 70.2)?
19. What is the output of print tuple[1:3] if tuple = ('abcd', 786 , 2.23, 'Tom', 70.2)?
20. What is the output of print tuple[2:] if tuple = ('abcd', 786 , 2.23, 'Tom', 70.2)?
21. What is the output of print smalltuple * 2 if smalltuple = ('Jack', 20)?

22. What is the output of print tuple + smalltuple if tuple = ('abcd', 786 , 2.23, 'Tom', 70.2) and smalltuple = ('Jack', 20)?
23. What do you mean by a Python dictionary?
24. How will you create a dictionary in Python?
25. How will you get all the keys from a dictionary?
26. How will you get all the values from a dictionary?
27. How will you convert a string to an int in Python?
28. How will you convert a string to a long in Python?
29. How will you convert a string to a float in Python?
30. How will you convert a object to a string in Python?
31. How will you convert a object to a regular expression in Python?
32. How will you convert a String to an object in Python?
33. How will you convert a string to a tuple in Python?
34. How will you convert a string to a list in Python?
35. How will you convert a string to a set in Python?
36. How will you create a dictionary using tuples in Python?
37. How will you convert a string to a frozen set in Python?
38. How will you convert an integer to a character in Python?
39. How will you convert an integer to a unicode character in Python?
40. How will you convert a single character to its integer value in Python?
41. How will you convert an integer to hexadecimal string in Python?
42. How will you convert an integer to octal string in Python?
43. How can you pick a random item from a list or tuple?
44. How can you pick a random item from a range?
45. How can you get a random number in python?
46. How will you set the starting value in generating random numbers?
47. How will you randomize the items of a list in place?
48. How will you capitalizes first letter of string?
49. How will you check in a string that all characters are alphanumeric?
50. How will you check in a string that all characters are digits?
51. How will you check in a string that all characters are in lowercase?
52. How will you check in a string that all characters are numbers?
53. How will you check in a string that it is properly title cased?
54. How will you check in a string that all characters are whitespaces?
55. How will you check in a string that all characters are in uppercase?
56. How will you merge elements in a sequence?
57. How will you get the length of the string?
58. How will you get a space-padded string with the original string left-justified to a total of width columns?
59. How will you convert a string to all lowercase?

60. How will you remove all leading whitespace in string?
61. How will you get the max alphabetical character from the string?
62. How will you get the min alphabetical character from the string?
63. How will you replaces all occurrences of old substring in string with new string?
64. How will you remove all leading and trailing whitespace in string?
65. How will you change case for all letters in string?
66. How will you get title cased version of string?
67. How will you convert a string to all uppercase?
68. How will you check in a string that all characters are decimal?
69. What is the difference between del() and remove() methods of list?
70. What is the output of len([1, 2, 3])?
71. What is the output of [1, 2, 3] + [4, 5, 6]?
72. What is the output of ['Hi!'] * 4?
73. What is the output of 3 in [1, 2, 3]?
74. What is the output of for x in [1, 2, 3]: print x?
75. What is the output of L[2] if L = [1,2,3]?
76. What is the output of L[-2] if L = [1,2,3]?
77. What is the output of L[1:] if L = [1,2,3]?
78. How will you get the length of a list?
79. How will you get the max valued item of a list?
80. How will you get the min valued item of a list?
81. How will you get the index of an object in a list?
82. How will you insert an object at given index in a list?
83. How will you remove last object from a list?
84. How will you remove an object from a list?
85. How will you reverse a list?
86. How will you sort a list?

❑❑❑