# 03 Introduction to Python

CONTENTS



Python was developed by Guido van Rossum at the National Research Institute for Mathematics and Computer Science in Netherlands during 1985-1990. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell and other scripting languages. Rossum was inspired by Monty Python's Flying Circus, a BBC comedy series and he wanted the name of his new language to be short, unique and mysterious. Hence he named it Python. It is a general-purpose interpreted, interactive, object-oriented and high-level Programming language. Python source code is available under the GNU General Public License (GPL) and it is now maintained by a core development team at the National Research Institute.

## 3.1 Features of Python

a) **Simple and easy-to-learn** – Python is a simple language with few keywords, simple structure and its syntax is also clearly defined. This makes Python a beginner's language.



b) **Interpreted and Interactive** - Python is processed at runtime by the interpreter. We need not compile the program before executing it. The Python prompt interact with the interpreter to interpret the programs that you have written. Python has an option namely interactive mode which allows interactive testing and debugging of code.

c) **Object-Oriented** – Python supports Object Oriented Programming (OOP) concepts that encapsulate code within objects. All concepts in OOPs like data hiding, operator overloading, inheritance etc. can be well written in Python. It supports functional as well as structured programming.

d) **Portable** - Python can run on a wide variety of hardware and software platforms and has the same interface on all platforms. All variants of Windows, Unix, Linux and Macintosh are to name a few.

e) **Scalable** - Python provides a better structure and support for large programs than shell scripting. It can be used as a scripting language or can be compiled to bytecode (intermediate code that is platform independent) for building large applications.

f) **Extendable** - You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient. It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

g) **Dynamic** - Python provides very high-level dynamic data types and supports dynamic type checking. It also supports automatic garbage collection.

h) **GUI Programming and Databases** - Python supports GUI applications that can be created and ported to many libraries and windows systems, such as Windows Microsoft Foundation Classes (MFC), Macintosh and the X Window system of Unix. Python also provides interfaces to all major commercial databases.

i) **Broad Standard Library** - Python's library is portable and cross platform compatible on UNIX, Linux, Windows and Macintosh. This helps in the support and development of a wide range of applications from simple text processing to browsers and complex games.

## 3.2 How to Run Python

There are three different ways to start Python.

**a) Using Interactive Interpreter**

You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window. Get into the command line of Python. For Unix/Linux, you can get into interactive mode by typing `$python or python%`. For Windows/Dos it is `C:>python`.

Invoking the interpreter without passing a script file as a parameter brings up the following prompt –

```
$ python
Python 2.7.10 (default, Sep 27 2015, 18:11:38)
[GCC 5.1.1 20150422 (Red Hat 5.1.1-1)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Type the following text at the Python prompt and press the Enter:

```
>>> print("Programming in Python!")
```

The result will be as given below.

```
Programming in Python!
>>>
```

**b) Script from the Command Line**

This method invokes the interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. A Python script can be executed at command line by invoking the interpreter on your application, as follows.

For Unix/Linux it is `$python script.py` or `python% script.py`. For Windows/Dos it is

```
C:>python script.py
```

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a `first.py` file.

```
print("Programming in Python!")
```

Now, try to run this program as follows.

```
$ python first.py
```

This produces the following result:

```
Programming in Python!
```

**c) Integrated Development Environment**

You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python. IDLE is the Integrated Development Environment (IDE) for UNIX and PythonWin is the first Windows interface for Python. All the programs in this book are tested in Python 3.4.3 for Windows.

## 3.3 Identifiers

A Python identifier is a name used to identify a variable, function, class, module or any other object. Python is case sensitive and hence uppercase and lowercase letters are considered distinct. The following are the rules for naming an identifier in Python.

a) Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example Total and total is different.

b) Reserved keywords (explained in section 3.4) cannot be used as an identifier.

c) Identifiers cannot begin with a digit. For example 2more, 3times etc. are invalid identifiers.

d) Special symbols like @, !, #, $, % etc. cannot be used in an identifier. For example sum@, #total are invalid identifiers.

e) Identifier can be of any length.

Some examples of valid identifiers are total, max_mark, count2, Student etc. Here are some naming conventions for Python identifiers.



- Class names start with an uppercase letter. All other identifiers start with a lowercase letter. Eg: Person
- Starting an identifier with a single leading underscore indicates that the identifier is private. Eg: _sum
- Starting an identifier with two leading underscores indicates a strongly private identifier. Eg: __sum
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name. foo__

## 3.4 Reserved Keywords

These are keywords reserved by the programming language and prevent the user or the programmer from using it as an identifier in a program. There are 33 keywords in Python 3.3. This number may vary with different versions. To retrieve the keywords in Python the following code can be given at the prompt. All keywords except True, False and None are in lowercase. The following list in Table 3.1 shows the Python keywords.

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

**Table 3.1: Python Keywords**

| | | | | |
|---|---|---|---|---|
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

## 3.5 Variables

Variables are reserved memory locations to store values. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Example Program

```
a = 100          # An integer assignment
b = 1000.0       # A floating point
name = "John"    # A string

print(a)
print(b)
print(name)
```

Here, 100, 1000.0 and "John" are the values assigned to a, b, and *name* variables, respectively. This produces the following result.

```
100
1000.0
John
```

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example

```
a, b, c = 1, 2, "Tom"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "Tom" is assigned to the variable c.

## 3.6 Comments in Python

Comments are very important while writing a program. It describes what the source code has done. Comments are for programmers for better understanding of a program. In Python, we use the hash (#) symbol to start writing a comment. A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment. It extends up to the newline character. Python interpreter ignores comment.

Example Program

```
>>>#This is demo of comment
>>>#Display Hello
>>>print("Hello")
```

This produces the following result. Hello

For multiline comments one way is to use (#) symbol at the beginning of each line. The following example shows a multiline comment. Another way of doing this is to use triple quotes, either ''' or """



Example 1

```
>>>#This is a very long sentence
>>>#and it ends after
>>>#Three lines
```

Example 2

```
>>>"""This is a very long sentence
>>>and it ends after
>>>Three lines"""
```

Both Example 1 and Example 2 given above produces the same result.

## 3.7 Indentation in Python

Most of the programming languages like C, C++ and Java use braces { } to define a block of code. Python uses indentation. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation can be decided by the programmer, but it must be consistent throughout the block. Generally four whitespaces are used for indentation and is preferred over tabspace. For example

```
if True:
    print("Correct")
else:
    print("Wrong")
```

But the following block generates error as indentation is not properly followed.

```
if True:
    print("Answer")
    print("Correct")
else:
    print("Answer")
  print("Wrong")
```

## 3.8 Multi-Line Statements

Instructions that a Python interpreter can execute are called statements. For example, a=1 is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed in coming Chapters. In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character(\). For example:

```
grand_total = first_item + \
              second_item + \
              third_item
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example

```
months = ['January', 'February', 'March', 'April',
          'May', 'June', 'July', 'August', 'September',
          'October', 'November', 'December']
```

We could also put multiple statements in a single line using semicolons, as follows.

```
a = 1; b = 2; c = 3
```

## 3.9 Multiple Statement Group (Suite)

A group of individual statements, which make a single code block is called a **suite** in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite. Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite. For example

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

## 3.10 Quotes in Python

Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals. The same type of quote should be used to start and end the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal.

```
word = 'single word'
sentence = "This is a short sentence."
paragraph = """This is a long paragraph. It consists of
          several lines and sentences."""
```

## 3.11 Input, Output and Import Functions

### 3.11.1 Displaying the Output

The function used to print output on a screen is the print statement where you can pass zero or more expressions separated by commas. The print function converts the expressions you pass into a string and writes the result to standard output.

**Example 1**

```
>>>print("Learning Python is fun and enjoy it.")
```



This will produce the following output on the screen.

```
Learning Python is fun and enjoy it.
```

**Example 2**

```
>>>a=2
>>>print("The value of a is",a)
```

This will produce the following output on the screen.

```
The value of a is 2
```

By default a space is added after the text and before the value of variable a. The syntax of *print* function is given below.

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Here, `objects` are the values to be printed. `Sep` is the separator used between the values. Space character is the default separator. `End` is printed after printing all the values. The default value for end is the new line. `File` is the object where the values are printed and its default value is `sys.stdout` (screen). Flush determines whether the output stream needs to be flushed for any waiting output. A True value forcibly flushes the stream. The following shows an example for the above syntax.

**Example**

```
>>>print(1,2,3,4)
>>>print(1,2,3,4,sep='+')
>>> print(1,2,3,4,sep='+',end='%')
```

The output will be as follows.

```
1 2 3 4
1+2+3+4
1+2+3+4%
```

The output can also be formatted to make it attractive according to the wish of a user. This will be discussed in the subsequent Chapters.

### 3.11.2 Reading the Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are **raw_input** and **input.**

`raw_input function`

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline). This prompts you to enter any string and it would display same string on the screen. raw_input function is not supported by Python 3.4.3 for Windows.

Example

```
str = raw_input("Enter your name: ");
print("Your name is : ", str)
```

Output

```
Enter your name: Collin Mark
Your name is : Collin Mark
```

**input function**

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

Example 1

```
n = input("Enter a number ");
print("The number is : ", n)
```

Output

```
Enter a number: 5
The number is : 5
```

Example 2

```
n = input("Enter an expression ");
print("The result is : ", n)
```

Output

```
Enter an expression: 5*2
The result is : 10
```

### 3.11.3 Import function

When the program grows bigger or when there are segments of code that is frequently used, it can be stored in different modules. A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension `.py`. Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the `import` keyword to do this. For example, we can import the `math` module by typing in `import math`.

```
>>> import math
>>> math.pi
3.141592653589793
```

## 3.12 Operators

Operators are the constructs which can manipulate the value of operands. Consider the expression a = b + c. Here a, b and c are called the operands and +, = are called the operators. There are different types of operators. The following are the types of operators supported by Python.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators



- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

### 3.12.1 Arithmetic Operators

Arithmetic operators are used for performing basic arithmetic operations. The following are the arithmetic operators supported by Python. Table 3.2 shows the arithmetic operators in Python.

**Table 3.2: Arithmetic Operators in Python**

| Operator | Operation | Description |
|---|---|---|
| + | Addition | Adds values on either side of the operator. |
| - | Subtraction | Subtracts right hand operand from left hand operand. |
| * | Multiplication | Multiplies values on either side of the operator. |
| / | Division | Divides left hand operand by right hand operand. |
| % | Modulus | Divides left hand operand by right hand operand and returns remainder. |
| ** | Exponent | Performs exponential (power) calculation on operators. |
| // | Floor Division | The division of operands where the result is the quotient in which the digits after the decimal point are removed. |

Example Program

```
a, b, c = 10, 5, 2
print("Sum=",(a+b))
print("Difference=",(a-b))
print("Product=",(a*b))
print("Quotient=",(a/b))
print("Remainder=",(b%c))
print("Exponent=",(b**2))
print("Floor Division=",(b//c))
```

Output

```
Sum= 15
Difference= 5
Product= 50
Quotient= 2
Remainder= 1
Exponent= 25
Floor Division= 2
```

### 3.12.2 Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. Python supports the following relational operators. Table 3.3 shows the comparison or relational operators in Python.

**Table 3.3: Comparison (Relational Operators) in Python**

| Operator | Description |
|---|---|
| == | If the values of two operands are equal, then the condition becomes true. |
| != | If values of two operands are not equal, then condition becomes true. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. |

Example Program

```
a,b=10,5
print("a==b is",(a==b))
print("a!=b is",(a!=b))
print("a>b is",(a>b))
print("a<b is",(a<b))
print("a>=b is",(a>=b))
print("a<=b is",(a<=b))
```

Output

```
a==b is False
a!=b is True
a>b is True
a<b is False
a>=b is True
a<=b is False
```

### 3.12.3 Assignment Operators

Python provides various assignment operators. Various shorthand operators for addition, subtraction, multiplication, division, modulus, exponent and floor division are also supported by Python. Table 3.4 provides the various assignment operators.



**Table 3.4: Assignment Operators**

| Operator | Description |
|---|---|
| = | Assigns values from right side operands to left side operand. |
| += | It adds right operand to the left operand and assign the result to left operand. |
| -= | It subtracts right operand from the left operand and assign the result to left operand. |
| *= | It multiplies right operand with the left operand and assign the result to left operand. |
| /= | It divides left operand with the right operand and assign the result to left operand. |
| %= | It takes modulus using two operands and assign the result to left operand. |
| **= | Performs exponential (power) calculation on operators and assign value to the left operand. |
| //= | It performs floor division on operators and assign value to the left operand. |

The assignment operator = is used to assign values or values of expressions to a variable. Example: a = b+ c. For example c+=a is equivalent to c=c+a. Similarly c-=a is equivalent to c=c-a, c*=a is equivalent to c=c*a, c/=a is equivalent to c=c/a, c%=a is equivalent to c=c%a, c**=a is equivalent to c=c**a and c//=a is equivalent to c=c//a.

Example Program

```
a,b=10,5
a+=b
print(a)
a,b=10,5
a-=b
print(a)
a,b=10,5
a*=b
print(a)
a,b=10,5
a/=b
print(a)
b,c=5,2
b%=c
print(b)
b,c=5,2
b**=c
print(b)
b,c=5,2
b//=c
print(b)
```

Output

```
15
5
60
2
1
25
2
```

### 3.12.4 Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. The following are the bitwise operators supported by Python. Table 3.5 gives a description of bitwise operators in Python.

**Table 3.5: Bitwise Operators**

| Operator | Operation | Description |
|---|---|---|
| & | Binary AND | Operator copies a bit to the result if it exists in both operands. |
| \| | Binary OR | It copies a bit if it exists in either operand. |
| ^ | Binary XOR | It copies the bit if it is set in one operand but not both. |
| ~ | Binary Ones Complement | It is unary and has the effect of 'flipping' bits. |
| << | Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. |
| >> | Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. |

Let *a* = 60 (0011 1100 in binary) and *b* = 2 (0000 0010 in binary)

Example Program

```
a,b=60,2
print(a&b)
print(a|b)
print(a^b)
print(~a)
print(a>>b)
print(a<<b)
```

Output

```
0
62
62
-61
15
240
```

Consider the first print statement `a&b`. Here a = 0011 1100

b = 0000 0010

When a bitwise and is performed, the result is 0000 0000. This is 0 in decimal. Similar is the case for all the print statements given above.



### 3.12.5 Logical Operators

Table 3.6 shows the various logical operators supported by Python language.

**Table 3.6: Logical Operators**

| Operator | Operation | Description |
|---|---|---|
| and | Logical AND | If both the operands are true then condition becomes true. |
| or | Logical OR | If any of the operands are true then condition becomes true. |
| not | Logical NOT | Used to reverse the logical state of its operand. |

Example Program

```
a,b,c,d=10,5,2,1
print((a>b) and (c>d))
print((a>b) or (d>c))
print(not(a>b))
```

Output

```
True
True
False
```

### 3.12.6 Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below in Table 3.7. Strings, lists and tuples will be covered in the forthcoming Chapter.

**Table 3.7: Membership Operators**

| Operator | Description |
|---|---|
| in | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. |

Example Program

```
s='abode'
print('a' in s)
print('f' in s)
print('f' not in s)
```

Output

```
True
False
True
```

### 3.12.7 Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators as shown in below Table 3.8.

**Table 3.8: Identity Operators**

| Operator | Description |
|---|---|
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. |

Example Program

```
a,b,c=10,10,5
print(a is b)
print(a is c)
print(a is not b)
```

Output

```
True
False
False
```

### 3.12.8 Operator Precedence

The following Table 3.9 lists all operators from highest precedence to lowest. Operator associativity determines the order of evaluation when they are of the some precedence and are not grouped by parenthesis. An operator may be left-associative or right-associative. In left-associative, the operator falling on the left side will be evaluated first, while in right-associative, operator falling on the right will be evaluated first. In Python '=' and '**' are right-associative while all other operators are left-associative.

**Table 3.9: Operator Precedence**

| Operator | Description |
|---|---|
| ** | Exponentiation (raise to the power) |
| ~ ,+, - | Complement, unary plus and minus |
| *, /, %, // | Multiply, divide, modulo and floor division |



| Operator | Description |
|---|---|
| +, - | Addition and subtraction |
| >>, << | Right and left bitwise shift |
| & | Bitwise 'AND' |
| ^ , \| | Bitwise exclusive 'OR' and regular 'OR' |
| <=, < > ,>= | Comparison operators |
| <>, ==, != | Equality operators |
| = ,%= ,/=, //= ,-= ,+=, *= ,**= | Assignment operators |
| is, is not | Identity operators |
| in, not in | Membership operators |
| not, or, and | Logical operators |

## 3.13 Conclusion

This Chapter gives an introduction to Python, its features, programming constructs like identifiers, reserved keywords, variables and various operators with illustrated examples. This Chapter also covers the purpose of indentation, the usage of comments, multi-line statements, multiple statements and quotes in Python, each one with illustrated examples. The basic Input/Output and import functions covered in this Chapter helps to handle the input, output and import operations in Python. Moreover how to execute Python is explained in this Chapter.

## 3.14 Review Questions

1. List some of the features of Python.
2. Is Python a case sensitive language?
3. How can you run Python?
4. Give the rules for naming an identifier in Python.
5. How can you give comments in Python?
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.

# 04 Data Types and Operations

CONTENTS



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

## 4.1 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)
```

### 4.1.1 Mathematical Functions

Python provides various built-in mathematical functions to perform mathematical calculations. The following Table 4.1 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 7.

**Table 4.1: 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. |

| Function | Description |
|---|---|
| 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: 0.693147180559453
```



```
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:
(0.0908759999999973, 12.0)
```

### 4.1.2 Trigonometric Functions

There are several built-in trigonometric functions in Python. The function names and its purpose are listed in Table 4.2.

**Table 4.2: Trigonometric Functions**

| Function | Description |
|---|---|
| sin(x) | Returns the sine of x in radians. |
| cos(x) | Returns the cosine of x in radians. |
| tan(x) | Returns the tangent of x in 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
```

### 4.1.3 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 4.3 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 4.3: 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
```

## 4.2 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 4.2.2.

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
```

### 4.2.1 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 4.4 shows the most commonly used escape characters and their description.

**Table 4.4: 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 |

### 4.2.2 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 4.5 shows various format symbols and their conversion.

**Table 4.5: 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" %(16,16))#usage
of %x
print("%X is the hexadecimal equivalent of %d" %(16,16))#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
```

### 4.2.3 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 character 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

```
16
-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

```
16
-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

```
16
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

```
16
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']
```

## 4.3 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 indices 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]
```

### 4.3.1 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.

### 4.3.2 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([func])** – 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([func])
list1 = [890, 147, 2.43, 100]
print("List before sorting:",list1)
list1.sort()
print("List after sorting:",list1)
```

Output

```
List before sorting: [890, 147, 2.43, 100]
List after sorting: [2.43, 100, 147, 890]
```

9. **list.pop(obj=list[-1])** – Removes or returns the last object obj from a list.



Example Program

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

Output

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

## 4.4 Tuple *Immutable*

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. It is not possible to remove individual items from a tuple. Eg. del tuple

#### 4.4.1 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')
```

### 4.5 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.

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}
```

#### 4.5.1 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
```

### 4.5.2 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
```

### 4.5.3 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})
```

## 4.6 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 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'}

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
```

Output

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, '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.

### 4.6.1 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'>
```

### 4.6.2 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']
```

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]
```

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') :",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}
```

## 4.7 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, string 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 string 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.

## 4.8 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 fromone data type to another. These functions return a new object representing the converted value. Table 4.6 shows various functions and their descriptions used for data type conversion.

**Table 4.6: 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. |



| Function | Description |
|---|---|
| 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'
sl={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", sl ,"to dictionary is ",dict(sl))
```

Output 2

```
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 3

```
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
```



## 4.9 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: python 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[0]))
```

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 hat 3 sat 5
List after shuffling:['3', '5', '1', 'Cat', 'sat', 'hat']
```

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

Program

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
```

```
sorted_d = sorted(d.items(), key=operator.itemgetter(1))
print('Dictionary in ascending order by value : ',sorted_d)
sorted_d = sorted(d.items(), key=operator.itemgetter(1),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 :  [(3, 4), (4, 3), (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}
```

## 4.10 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. The review questions given in the Section 4.11 are the most frequently asked questions for Interviews and University Examinations. This help students to prepare for University Examinations, Certification Examinations and Job Interviews.



## 4.11 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 randomizes 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 numerics?
53. How will you check in a string that all characters are whitespaces?
54. How will you check in a string that it is properly titlecased?
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 with 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 titlecased 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?

# 05 Flow Control

CONTENTS



## 5.1 Decision Making

Decision making is required when we want to execute a code only if a certain condition is satisfied. The `if...elif...else` statement is used in Python for decision making.

### 5.1.1 *if* statement

**Syntax**

```
if test expression:
    statement(s)
```

The following Fig. 5.1 shows the flowchart of the `if` statement.



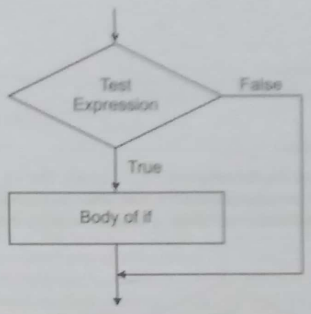


**Fig. 5.1: Flowchart of *if* Statement**

Here, the Program evaluates the `test expression` and will execute statement(s) only if the text expression is `True`. If the text expression is `False`, the statement(s) is not executed. In Python, the body of the `if` statement is indicated by the indentation. Body starts with an indentation and ends with the first unindented line. Python interprets non-zero values as `True`. `None` and `0` are interpreted as `False`.

Example Program

```
num =int(input("Enter a number: "))
if num == 0:
    print("Zero")
print("This is always printed")
```

Output 1

```
Enter a number: 0
Zero
This is always printed
```

Output 2

```
Enter a number: 2
This is always printed
```

In the above example, `num == 0` is the test expression. The body of `if` is executed only if this evaluates to `True`. When user enters 0, test expression is True and body inside `if` is executed. When user enters 2, test expression is False and body inside `if` is skipped. The statement `This is always printed` because the `print` statement falls outside the `if` block. The statement outside the `if` block is shown by unindentation. Hence, it is executed regardless of the test expression or it is always executed.

### 5.1.2 *if...else* statement

Syntax

```
if test expression:
    Body of if
else:
    Body of else
```

The following Fig. 5.2 shows the flowchart of `if...else`. The `if..else` statement evaluates `test expression` and will execute body of `if` only when test condition is `True`. If the condition is `False`, body of `else` is executed. Indentation is used to separate the blocks.

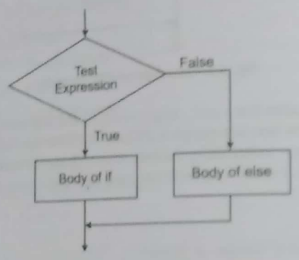


**Fig. 5.2: Flow chart of `if... .else`**

Example Program

```
num =int(input("Enter a number: "))
if num >= 0:
    print("Positive Number or Zero")
else:
    print("Negative Number")
```

Output 1

```
Enter a number: 5
Positive Number or Zero
```

Output 2

```
Enter a number: -2
Negative Number
```

In the above example, when user enters 5, the test expression is `True`. Hence the body of `if` is executed and body of `else` is skipped. When user enters -2, the test expression is `False` and body of `else` is executed and body of `if` is skipped.



### 5.1.3 *if...elif...else* statement

Syntax

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The `elif` is short for else if. It allows us to check for multiple expressions. If the condition for `if` is `False`, it checks the condition of the next `elif` block and so on. If all the conditions are `False`, body of else is executed. Only one block among the several `if...elif...else` blocks is executed according to the condition. A `if` block can have only one `else` block. But it can have multiple `elif` blocks. The following Fig. 5.3 shows the flow chart for `if..elif..else` statement.

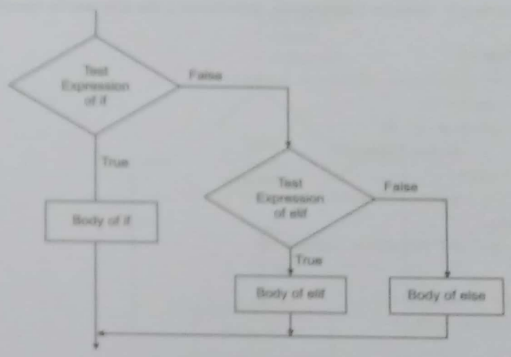


**Fig. 5.3: Flow chart for `if..elif..else`**

Example Program

```
num =int(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive Number
```

Output 2

```
Enter a number: 0
Zero
```

Output 3

```
Enter a number: -2
Negative Number
```

### 5.1.4 Nested *if* statement

We can have a `if...elif...else` statement inside another `if...elif...else` statement. This is called nesting in computer programming. Indentation is the only way to identify the level of nesting.

Example Program

```
num = int(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive Number
```

Output 2

```
Enter a number: 0
Zero
```

Output 3

```
Enter a number: -2
Negative Number
```

## 5.2 Loops

Generally, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on. There will be situations when we need to execute a block of code



several number of times. Python provides various control structures that allow for repeated execution. A loop statement allows us to execute a statement or group of statements multiple times.

### 5.2.1 *for* loop

The `for` loop in Python is used to iterate over a sequence (list, tuple, string) or other objects that can be iterated. Iterating over a sequence is called traversal. Fig. 5.4 shows the flow chart of `for` loop.

**Syntax**

```
for item in sequence:
    Body of for
```

Here, `item` is the variable that takes the value of the item inside the sequence of each iteration. The sequence can be list, tuple, string, set etc. Loop continues until we reach the last item in the sequence. The body of `for` loop is separated from the rest of the code using indentation.

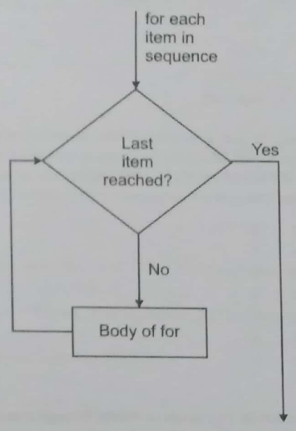


**Fig. 5.4: Flow chart of `for` loop**

Example Program 1

```
#Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [2,4,6,8,10]
# variable to store the sum
```

```
sum = 0
# iterate over the list
for item in numbers:
    sum = sum+item
# print the sum
print("The sum is",sum)
```

Output 1

```
The sum is 30
```

Example Program 2

```
flowers = ['rose', 'lotus', 'jasmine']
for flower in flowers:
    print('Current flower :', flower)
```

Output 2

```
Current flower : rose
Current flower : lotus
Current flower : jasmine
```

Example Program 3

```
for letter in 'Program':
    print('Current letter :', letter)
```

Output 3

```
Current letter : P
Current letter : r
Current letter : o
Current letter : g
Current letter : r
Current letter : a
Current letter : m
```

**range() function**

We can use the `range()` function in `for` loops to iterate through a sequence of numbers. It can be combined with the `len()` function to iterate though a sequence using indexing. `len()` function is used to find the length of a string or number of elements in a list, tuple, set etc.

Example Program

```
flowers = ['rose', 'lotus', 'jasmine']
for i in range(len(flowers)):
    print('Current flower :', flowers[i])
```



Output

```
Current flower : rose
Current flower : lotus
Current flower : jasmine
```

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers). We can also define the `start`, `stop` and `step_size` as `range(start,stop,step_size)`. The default value of `step_size` is 1, if not provided. This function does not store all the values in memory. It keeps track of the the `start`, `stop`, `step_size` and generates the next number.

Example Program

```
for num in range(2, 10, 2):print("Number = ", num)
```

Output

```
Number = 2
Number = 4
Number = 6
Number = 8
```

### 5.2.2 *for* loop with *else*

Python supports to have an else statement associated with a loop statement. If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has finished iterating the list. A `break` statement can be used to stop a `for` loop. In this case, the `else` part is ignored. Hence, a `for` loop's `else` part runs if no break occurs. Break statement is discussed in the Section 5.4 Control Statements.

Example Program

```
#Program to find whether an item is present in the list
list=[10,12,13,34,27,98]
num = int(input("Enter the number to be searched in the list:
"))
for item in range(len(list)):
 if list[item] == num:
   print("Item found at: ",(item+1))
   break
else:
   print("Item not found in the list")
```

Output 1

```
Enter the number to be searched in the list:34
Item found at:4
```

Output 2

```
Enter the number to be searched in the list:99
Item not found in the list
```

### 5.2.3 *while* loop

The `while` loop in Python is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know the number of times to iterate in advance. Fig. 5.5 shows the flow chart of a `while` loop.

**Syntax**

```
while test_expression:
    Body of while
```

In `while` loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process is continued until the `test_expression` evaluates to `False`. In Python, the body of the `while` loop is determined through indentation. Body starts with indentation and the first unindented line shows the end. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

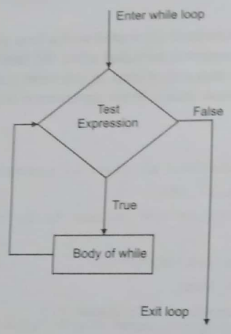


**Fig. 5.5: Flow chart for while loop**

Example Program

```
# Program to find the sum of first N natural numbers
n =int(input("Enter the limit: "))
sum=0
i=1
while i<=n:
    sum=sum+i
    i=i+1
print("Sum of first", n, "natural numbers is ", sum)
```

Output

```
Enter the limit: 5
Sum of first 5 natural numbers is 15
```

**Illustration**

In the above program, when the text `Enter the limit` appears, 5 is given as the input. ie. n=5. Initially, a counter i is set to 1 and sum is set to 0. When the condition is checked for the first time i<=n, it is True. Hence the execution enters the body of while loop. Sum is added with i and i is incremented. Since the next statement is unindented, it assumes the end of while block. This process is repeated when the condition given in the while loop is False and the control goes to the next print statement to print the sum.

### 5.2.4 *while* loop with *else* statement

If the `else` statement is used with a `while` loop, the `else` statement is executed when the condition becomes False. `while` loop can be terminated with a `break` statement. In such case, the `else` part is ignored. Hence, a `while` loop's `else` part runs if no break occurs and the condition is False.

Example Program 1

```
# Demo of While with else
count=1
while count<=3:
    print("Python Programming")
    count=count+1
else:
    print("Exit")
print("End of Program")
```

Output 1

```
Python Programming
Python Programming
Python Programming
Exit
End of Program
```

**Illustration**

In this example, initially the count is 1. The condition in the while loop is checked and since it is True, the body of the while loop is executed. The loop is terminated when the condition is False and it

goes to the else statement. After executing the else block it will move on to the rest of the Program statements. In this example the print statement after the else block is executed.

Example Program 2

```
# Demo of While with else
count=1
while count<=3:
    print("Python Programming")
    count=count+1
    if count==2:break
else:
    print("Exit")
print("End of Program")
```

Output 2

```
Python Programming
End of Program
```

**Illustration**

In this example, initially the count is 1. The condition in the `while` loop is checked and since it is True, the body of the `while` loop is executed. When the `if` condition inside the `while` loop is True, ie. when count becomes 2, the control is exited from the `while` loop. It will not execute the `else` part of the loop. The control will moves on to the next statement after the `else`. Hence the print statement `End of Program` is executed.

## 5.3 Nested Loops

Sometimes we need to place a loop inside another loop. This is called nested loop. We can have nested loops for both `while` and `for`.

Syntax for nested for loop

```
for iterating_variable in sequence:
    for iterating_variable in sequence:
        statements(s)
    statements(s)
```

Syntax for nested while loop

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

Q: Write a program to print the prime numbers starting from 1 upto a limit entered by the user.



Program using **nested `for`**

```
#Program to generate prime numbers between 2 Limits
import math
n =int(input("Enter a Limit:"))
for i in range(1,n):
  k=int(math.sqrt(i))
  for j in range(2,k+1):
   if i%j==0:break
  else: print(i)
```

Output

```
Enter a Limit:12
1
2
3
5
7
11
```

The limit entered by the user is stored in n. To find whether a number is prime, the logic used is to divide that number from 2 to square root of that number (Since all numbers are completely divisible by 1, we started from 2). If the remainder of this division is zero at any time, that number is skipped and moved to next number. A complete division shows that the number is not prime. If the remainder is not zero at any time, it shows that it is a prime number.

The outer `for` loop starts from 1 to the input entered by the user. Initially i=1. A variable k is used to store the square root of the number. The inner `for` loop is used to find whether the number is completely divisible by any number between 2 and square root of that number(k). If it is completely divisible by any number between 2 and k, the number is not prime, else the number is considered prime. The same steps are repeated until the limit entered by the user is reached.

The above program is rewritten using nested while and is given below.

Program using **nested `while`**

```
#Program to generate prime numbers between 2 Limits
import math
n =int(input("Enter a Limit:"))
i=1
while i<=n:
  k=int(math.sqrt(i))
  j=2
  while j<=k:
```

```
    if i%j==0:break
    j+=1
  else: print(i)
  i+=1
```

Output

```
Enter a Limit:12
1
2
3
5
7
11
```

## 5.4 Control Statements

Control statements change the execution from normal sequence. Loops iterate over a block of code until test expression is False, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. The `break` and `continue` statements are used in these cases. Python supports the following three control statements.

1. `break`
2. `continue`
3. `pass`

### 5.4.1 `break` statement

The `break` statement terminates the loop containing it. Control of the Program flows to the statement immediately after the body of the loop. If it is inside a nested loop (loop inside another loop), `break` will terminate the innermost loop. It can be used with both `for` and `while` loops. Fig. 5.6 shows the flow chart of `break` statement.

Example Program 1

```
#Demo of break statement in Python
for i in range(2,10,2):
 if i==6: break
 print(i)
print("End of Program")
```

Output 1

```
2
4
End of Program
```



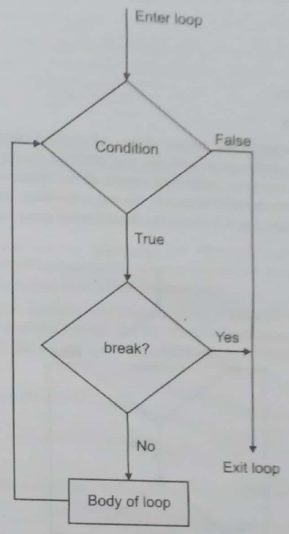


**Fig 5.6: Flow chart of `break` statement**

**Illustration**

In this the `for` loop is intended to print the even numbers from 2 to 10. The `if` condition checks whether i=6. If it is 6, the control goes to the next statement after the `for` loop. Hence it goes to the print statement `End of Program`.

Example Program 2

```
#Demo of break statement in Python
i=5
while i>0:
 if i==3: break
 print(i)
 i=i-1
print("End of Program")
```

Output 2
3
4
End of Program

### 5.4.2 `continue` statement

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration. Continue returns the control to the beginning of the loop. The `continue` statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The **continue** statement can be used in both *while* and *for* loops. Fig. 5.7 shows the flow chart of `continue` statement.

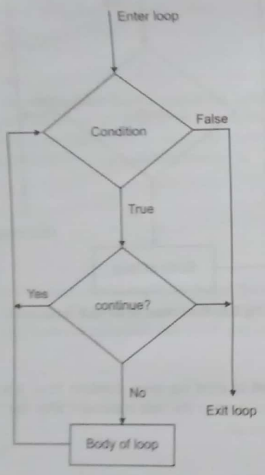


Fig. 5.7: Flow chart of `continue`

Example Program

```
# Demo of continue in Python
for letter in 'abcd':
 if letter =='c': continue
 print(letter)
```



Output
a
b
d

**Illustration**

When the letter=c, the continue statement is executed and the control goes to the beginning of the loop, bypasses the rest of the statements in the body of the loop. In the output the letters from "a" to "d" except "c" gets printed.

### 5.4.3 *pass* statement

In Python Programming, `pass` is a null statement. The difference between a comment and `pass` statement in Python is that, while the interpreter ignores a comment entirely, `pass` is not ignored. But nothing happens when it is executed. It results in no operation.

It is used as a placeholder. Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. The function or loop cannot have an empty body. The interpreter will not allow this. So, we use the `pass` statement to construct a body that does nothing.

Example

```
for val in sequence:
    pass
```

## 5.5 Types of Loops

There are different types of loops depending on the position at which condition is checked.

### 5.5.1 Infinite Loop

A loop becomes infinite loop if a condition never becomes `False`. This results in a loop that never ends. Such a loop is called an infinite loop. We can create an infinite loop using `while` statement. If the condition of `while` loop is always `True`, we get an infinite loop. We must use `while` loops with caution because of the possibility that the condition may never resolves to a `False` value.

Example Program

```
count=1
while count==1:
  n=input("Enter a Number:")
  print("Number=", n)
```

This will continue running unless you give CTRL+C to exit from the loop.

### 5.5.2 Loops with condition at the top

This is a normal `while` loop without `break` statements. The condition of the `while` loop is at the top and the loop terminates when this condition is `False`. This loop is already explained in Section 5.2.3

### 5.5.3 Loop with condition in the middle

This kind of loop can be implemented using an infinite loop along with a conditional break in between the body of the loop. Fig. 5.8 shows the flow chart of a loop with condition in the middle.

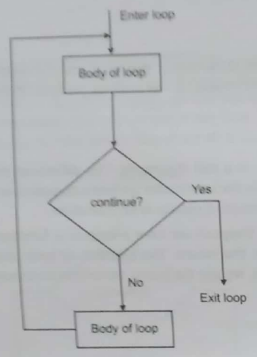


**Fig. 5.8: Loop with condition in the middle**

Example Program

```
vowels='aeiou'
# Infinite Loop
while True:
    v = input ("Enter a letter:")
    if v in vowels:
        print(v, "is a vowel")
        break
    print("This is not a vowel, Enter another letter")
print("End of Program")
```

Output

```
Enter a letter:t
This is not a vowel, Enter another letter
Enter a letter:o
o is a vowel
End of Program
```



### 5.5.4 Loop with condition at the bottom

This kind of loop ensures that the body of the loop is executed at least once. It can be implemented using an infinite loop along with a conditional break at the end. This is similar to the `do...while` loop in C. Fig. 5.9 shows the flow chart of loop with condition at the bottom.

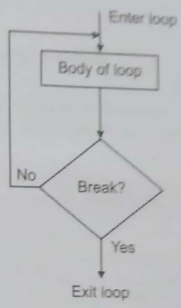


**Fig 5.9: Loop with condition at the bottom**

Example Program

```
# Demo of loop with condition at the bottom
choice=0
a,b=6,3
while choice !=5:
 print("Menu")
 print("1. Addition")
 print("2. Subtraction")
 print("3. Multiplication")
 print("4. Division")
 print("5. Exit")
 choice =int(input("Enter your choice:"))
 if choice==1: print("Sum=",(a+b))
 if choice==2: print("Difference=", (a-b))
 if choice==3: print("Product=",(a*b))
 if choice==4: print("Quotient=", (a/b))
 if choice==5: break
print("End of Program")
```

Output

```
Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice:2
Difference= 3
Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice:5
End of Program
```

## 5.6 Solved Lab Exercises

1. Program to find the GCD of 2 numbers.

Program

```
# GCD of two numbers
a =int(input("Enter first number:")) #Reads First Number
b =int(input ("Enter second number:")) #Reads Second Number
for i in range (1, min(a,b)+1):
 if a%i==0 and b%i==0: gcd=i
print("GCD of",a ,"and",b, "is",gcd)
```

Output

```
Enter first number:12
Enter second number:18
6
```

2. Program to find the factorial of a number.

Program

```
# Factorial of a number
a =int(input("Enter the number:")) #Reads Number
fact=1
```



```
for i in range (1, a+1):
 fact=fact*i
print("Factorial of", a, "is", fact)
```

Output

```
Enter the number:5
Factorial of 5 is 120
```

3. Program to generate fibonacci series of N terms.

Program

```
# Fibonacci series of first N numbers
n =int(input("Enter the number of terms:")) #Reads the limit
f1,f2=0,1
f3=f1+f2
print("Fibonacci series of first", n, "terms")
print(f1)
print(f2)
for i in range (3, n+1):
 print(f3)
 f1=f2
 f2=f3
 f3=f1+f2
```

Output

```
Enter the number of terms:5
Fibonacci series of first 5 terms
0
1
1
2
3
```

4. Program to count the number of vowels.

Program

```
# Count the number of vowels
s = input("Enter a String:")#Reads the String
count=0
for i in s:
 if i in 'aeiouAEIOU':
   count+=1
print("The number of vowels in ", s, "is", count)
```

Output

```
Enter a String:Python Programming
The number of vowels in Python Programming is 4
```

5. Program to find the sum of all items in a list.

Program

```
# Sum of all elements in a list
list = input("Enter a list:")#Reads the List from the keyboard
list1=map(int,list.split())# Convert the items in it to integers
sum=0 #initialize sum to 0
for i in list1:sum+=i
print("The sum of all items in list", list, "is", sum)
```

Output

```
Enter a list:1 2 3 4
The sum of all items in list 1 2 3 4 is 10
```

6. Write a program that prints the numbers from 1 to 20. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz". This is famously known as the FizzBuzz test.

Program

```
#Program for FizzBuzz Test
for i in range(1,20):
    if i%3==0 and i%5==0: print("FizzBuzz")
    elif i%3==0:print("Fizz")
    elif i%5==0:print("Buzz")
    else: print(i)
```

Output

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
```



```
Fizz
13
14
FizzBuzz
16
17
Fizz
19
```

7. Write a program that prints the following pyramid on the screen. The number of lines must be obtained from the user as input.

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

Program

```
#Program for Pyramid
n=int(input("Enter the step size:"))
for i in range(1,n+1):
    for j in range(1,i+1):
        print(i,end=' ')
    print()
```

Output

```
Enter the step size:5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

8. Write a program to find Primitive Pythagorean Triads A pythagorean triad has the property $a^2 + b^2 = c^2$. By primitive we mean triads that do not 'depend' on others. For example, (4,3,5) is a variant of (3,4,5) and hence is not primitive. And (10,24,26) is easily derived from (5,12,13) and should not be displayed by our Program. Write a Program to print primitive pythagorean triads. The Program should generate all triads with a, b values in the range 0—50.

Program

```
#Pythagorean Triad
for i in range(1,50):
    for j in range(1,i):
```

```
        for k in range(i,j):
          if k*k+j*j==i*i:
            flag=0
            for i in range(2,i):
                if i%1==0 and j%1==0 and k%1==0:
                    flag=1
                    break
            if flag:continue
            print("a=",k,"b=",j,"c=",i)
```

Output

```
a= 3 b= 4 c= 5
a= 5 b= 12 c= 13
a= 8 b= 15 c= 17
a= 7 b= 24 c= 25
a= 20 b= 21 c= 29
a= 12 b= 35 c= 37
a= 9 b= 40 c= 41
a= 28 b= 45 c= 53
a= 11 b= 60 c= 61
a= 33 b= 56 c= 65
a= 16 b= 63 c= 65
a= 48 b= 55 c= 73
a= 36 b= 77 c= 85
a= 13 b= 84 c= 85
a= 39 b= 80 c= 89
a= 65 b= 72 c= 97
```

9. Write a program that generates a list of all four digit numbers that have all their digits even and are perfect squares. For example, the output should include 6400 but not 8100 (one digit is odd) or 4248 (not a perfect square).

Program

```
#Four digit perfect square with even digits
import math
for i in range(1000,10000):
    num=int(math.sqrt(i))
    if num*num==i:
        n=i
        while n!=0:
            r=n%10
```



```
            n=n//10
            if r%2!=0:break
        else:print(i)
```

Output

```
4624
6084
6400
8464
```

10. Write a program to display the following pyramid. The number of lines has to be a parameter obtained from the user. The pyramid must appear aligned to the left edge of the screen.

```
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
```

Program

```
#Program for Pyramid
n=int(input("Enter the step size:"))
for i in range(1,n+1):
    k=i
    for j in range(1,i+1):
        print(k,end=' ')
        k+=i
    print()
```

Output

```
Enter the step size:5
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
```

11. Write a program to display the following output. The last number where the Program will stop printing has to be a parameter obtained from the user. The pyramid must appear aligned to the left edge of the screen. Note that depending on the last number, the base of the pyramid may be smaller than the line above it.

```
1
2 3
```

```
4 5 6
7 8 9 10
11 12
```

Program

```
#Program for Pyramid
n=int(input("Enter the limit for the pyramid:"))
k=1
for i in range(1,n+1):
        for j in range(1,i+1):
          if k==n+1: break
          print(k,end=' ')
          k+=1
        print()
        if k==n+1: break
```

Output

```
Enter the limit for the pyramid:12
1
2 3
4 5 6
7 8 9 10
11 12
```

12. Given an empty chessboard and one Bishop placed in any square, say (r, c), generate the list of all squares the Bishop could move to.

Program

```
# Movement of Bishop in Chess
r = int(input("Enter row number:"))
c = int(input("Enter column number:"))
while (c in range(1,9) and r in range(1,9)):
        print("Possible Movements in (row, col)")
        if r==1 and c==1:
         print(r,c+1)
         print(r+1,c)
         break
        elif r==1 and c==8:
         print(r,c-1)
         print(r+1,c)
```



```
         break
        elif c==1 and r==8:
         print(r-1,c)
         print(r,c+1)
         break
        elif c==8 and r==8:
         print(r-1,c)
         print(r,c-1)
         break
        elif c==1 and r<8:
         print(r,c+1)
         print(r+1,c)
         print(r-1,c)
         break
        elif r==1:
         print(r+1,c)
         print(r,c+1)
         print(r,c-1)
         break
        elif c==8:
         print(r-1,c)
         print(r+1,c)
         print(r,c-1)
         break
        elif r==8:
         print(r-1,c)
         print(r, c+1)
         print(r,c-1)
         break
        else:
          print(r,c-1)
          print(r, c+1)
          print(r+1, c)
          print(r-1, c)
          break
else:
        print("Invalid range for row or column")
```

Output

```
Enter row number:3
Enter column number:4
Possible Movements in (row, col)
3 3
3 5
4 4
2 4
```

13. Write a Python program which accept the user's name and print them in reverse order with a space between them.

Program

```
name = input("Input your Name : ")
l=name.split()
l.reverse()
print("Reversed name:",end=' ')
for i in l:
print(i,end=' ')
```

Output

```
Input your Name : Sam Charles Alexander
Reversed name: Alexander Charles Sam
```

14. Write a Python program to count a number in a given list.

Program

```
str=input("Enter a list(values space separated):")
lis=list(map(int,str.split()))
n=int(input("Enter the number to search for the number of
occurrences:"))
print(lis)
print("Number of occurrences of",n,"is",lis.count(n),"times")
```

Output

```
Enter a list(values space separated):1 4 5 2 3 5 2 5 2 6 2
Enter the number to search for the number of occurrences:2
[1, 4, 5, 2, 3, 5, 2, 5, 2, 6, 2]
Number of occurrences of 2 is 4 times
```

15. Write a Python program to get the n (non-negative integer) copies of the first 2 characters of a given string. Print n copies of the whole string if the length is less than 2.



Program

```
str1=input("Enter a String:")
n=int(input("Enter the number of copies of first two
characters:"))
flen = 2
if flen > len(str1):
    flen = len(str1)
substr = str1[:flen]
result = ""
for i in range(n):
      result = result + substr
print("Copy of the substring:",result)
```

Output

```
Enter a String:Python Programming
Enter the number of copies of first two characters:4
Copy of the substring: PyPyPyPy
```

16. Write a Python program to check whether a specified value is contained in a group of values.

Program

```
lis=input("Enter a list(values space separated):")
n=int(input("Enter the number to be searched:"))
lis1=list(map(int,lis.split()))
print(lis1)
for value in lis1:
  if n == value:

    print("The number", n ,"is found in the list.")
    break
else:
    print("The number", n ,"is not found in the list")
```

Output

```
Enter a list(values space separated):1 2 3
Enter the number to be searched:4
[1, 2, 3]
The number 4 is not found in the list
```

17. Write a Python program to concatenate all elements in a list into a string and return it.

Program

```
lis=input("Enter a list(space separated):")
sl=lis.split()
print(sl)
result= ''
for element in sl:
  result += str(element)
print("Concatenated elements in the list:",result)
```

Output

```
Enter a list(space separated):1 2 3 4
['1', '2', '3', '4']
Concatenated elements in the list: 1234
```

18. Write a Python program to print all even numbers from a given numbers list in the same order and stop the printing if any numbers that come after 237 in the sequence.

Program

```
lis=input("Enter a list(elements space separated):")
lisl=list(map(int,lis.split()))
print(lisl)
print("Even Numbers upto 237")
for x in lisl:
    if x == 237:
        break
    elif x % 2 == 0:
        print(x, end=' ')
```

Output

```
Enter a list(elements space separated):12 3 34 21 56 78 21 90
237 12 23 45
[12, 3, 34, 21, 56, 78, 21, 90, 237, 12, 23, 45]
Even Numbers upto 237
12 34 56 78 90
```

19. Write a Python program to get the least common multiple (LCM) of two positive integers.

Program

```
x=int(input("Enter first positive integer:"))
y=int(input("Enter second positive integer:"))
if x > y:
    z = x
else:
    z = y
while True:
    if (z % x == 0) and (z % y == 0):
        lcm = z
        break
    z += 1
print("LCM of",x, "and",y,"is",lcm)
```



Output

```
Enter first positive integer:6
Enter second positive integer:10
LCM of 6 and 10 is 30
```

20. Write a Python program to count the number of characters (character frequency) in a string.

Program

```
dict = {}
str1=input("Enter a string:")
for n in str1:
    if n in dict:
        dict[n] += 1
    else:
        dict[n] = 1
print("Character frequency")
for k,v in dict.items():
    print(k,v)
```

Output

```
Enter a string:pythonprogramming
Character frequency
r 2
m 2
h 1
n 2
t 1
i 1
a 1
y 1
```

```
q 2
p 2
o 2
```

21. Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead the empty string.

Program

```
str1=input("Enter a string:")
if len(str1) < 2:
    print(None)
else:
    print("String made from last two characters of both
ends:",(str1[0:2] + str1[-2:]))
```

Output

```
Enter a string:Python Programming
String made from last two characters of both ends: Pyng
```

22. Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string is already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

Program

```
str1=input("Enter a String:")
length = len(str1)
if length > 2:
    if str1[-3:] == 'ing':
      str1 += 'ly'
    else:
      str1 += 'ing'
print("New String:",str1)
```

Output

```
Enter a String:Python Programm
New String: Python Programming
```

23. Write a Python program to find the first appearance of the substring 'not' and 'poor' from a given string, if 'not' follows the 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.

Program

```
str1=input("Enter a string:")
snot = str1.find('not')
sbad = str1.find('poor')
if sbad > snot:
    str1 = str1.replace(str1[snot:(sbad+4)], 'good')
print(str1)
```



Output

```
Enter a string:It is not that poor
It is good
```

24. Write a Python function that takes a list of words and returns the length of the longest one.

Program

```
lis=input("Enter a list with some strings (space separated):")
words_list=lis.split()
word_len = []
for n in words_list:
    word_len.append((len(n), n))
    print(word_len)
    word_len.sort()
    print(word_len)
print("Longest Word:",word_len[-1][1])
```

Output

```
Enter a list with some strings (space separated):apple orange
pear kiwi
[(5, 'apple')]
[(5, 'apple')]
[(5, 'apple'), (6, 'orange')]
[(5, 'apple'), (6, 'orange')]
[(5, 'apple'), (6, 'orange'), (4, 'pear')]
[(4, 'pear'), (5, 'apple'), (6, 'orange')]
[(4, 'pear'), (5, 'apple'), (6, 'orange'), (4, 'kiwi')]
[(4, 'kiwi'), (4, 'pear'), (5, 'apple'), (6, 'orange')]
Longest Word: orange
```

25. Write a Python program to remove the characters which have odd index values of a given string.

Program

```
str1=input("Enter a String:")
result = ""
for i in range(0,len(str1),2):
    if i % 2 == 0:
      result = result + str1[i]
```

```
print("String after removing characters in odd
positions:",result)
```

Output

```
Enter a String:Python programming
String after removing characters in odd positions: Pto rgamn
```

26. Write a Python program to count the occurrences of each word in a given sentence.

Program

```
strl=input("Enter a String:")
counts = {}
words = strl.split()
for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
for k,v in counts.items():
    print(k,v)
```

Output

```
Enter a String:Learning Python is fun and Python is powerful
powerful 1
Python 2
fun 1
and 1
is 2
Learning 1
```

27. Write a Python program that accepts a comma separated sequence of words as input and prints the unique words in sorted form (alphanumerically).

Program

```
items = input("Input comma separated sequence of words:")
words=items.split(",")
for word in words:
  lis=(",".join(sorted(list(set(words)))))
print(lis)
```

Output

```
Input comma separated sequence of words:yellow,white,blue,red,gr
een,blue,red,black
black,blue,green,red,white,yellow
```



28. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

Program

```
lis=input("Enter a list(space separated):")
words=list(lis.split())
ctr = 0
for word in words:
    if len(word) > 1 and word[0] == word[-1]:
      ctr += 1
print("Count=",ctr)
```

Output

```
Enter a list(space separated):cat bob pop dog am pat abba
Count= 3
```

29. Write a Python program to remove duplicates from a list.

Program

```
lis=input("Enter a list(space separated):")
lis1=list(lis.split())
uniq_items = []
for x in lis1:
    if x not in uniq_items:
        uniq_items.append(x)
print(uniq_items)
```

Output

```
Enter a list(space separated):1 2 bat 12 bat 3 1 2
['1', '2', 'bat', '12', '3']
```

30. Write a Python program to check a list is empty or not.

Program

```
l=input("Enter a list(space separated):")
lis =list(l.split())
if not lis:
  print("List is empty")
else:
  print("List is non-empty")
  print(lis)
```

Output

```
Enter a list(space separated):1 2 3 4 5
List is non-empty
['1', '2', '3', '4', '5']
```

31. Write a Python program to find the list of words that are longer than n from a given list of words.

Program

```
str=input("Enter a list of words(space separated):")
n=int(input("Enter a length:"))
txt=str.split()
word_len=[]
for x in txt:
    if len(x) > n:
        word_len.append(x)
print("Words with length greater than", n,"=",word_len)
```

Output

```
Enter a list of words(space separated):bat apple mango rabbit
rat
Enter a length:3
Words with length greater than 3 = ['apple', 'mango', 'rabbit']
```

32. Write a Python program to print a specified list after removing the 0th, 2nd, 4th and 5th elements.

Program

```
list1=input("Enter a list(space separated):")
lis1=list(list1.split())
lis1= [x for (i,x) in enumerate(lis1) if i not in (0,2,4,5)]
print(lis1)
```

Output

```
Enter a list(space separated):mango banana orange kiwi pear
apple
['banana', 'kiwi']
```

33. Write a Python program to generate a 3*4*6 3D array whose each element is *.

Program

```
array = [[ ['*' for col in range(6)] for col in range(4)] for
row in range(3)]
print(array)
```



Output

```
[[['*', '*', '*', '*', '*', '*'], ['*', '*', '*', '*', '*',
'*'], ['*', '*', '*', '*', '*', '*'], ['*', '*', '*', '*', '*',
'*']], [['*', '*', '*', '*', '*', '*'], ['*', '*', '*', '*', '*',
'*', '*'], ['*', '*', '*', '*', '*', '*'], ['*', '*', '*', '*',
'*', '*']], [['*', '*', '*', '*', '*', '*'], ['*', '*', '*', '*',
'*', '*', '*'], ['*', '*', '*', '*', '*', '*'], ['*', '*', '*',
'*', '*', '*']]]
```

34. Write a Python program to print the numbers of a specified list after removing even numbers from it.

Program

```
num1 =input("Enter an integer list(space separated):")
num =list(map(int,num1.split()))
num=[x for x in num if x%2!=0]
print("List after removing even numbers",end=' ')
print(num)
```

Output

```
Enter an integer list(space separated):1 2 4 21 4 56 3
List after removing even numbers [1, 21, 3]
```

35. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).

Program

```
l = list()
for i in range(1,21):
      l.append(i**2)
print(l[:5])
print(l[-5:])
```

Output

```
[1, 4, 9, 16, 25]
[256, 289, 324, 361, 400]
```

36. Write a Python script to concatenate following dictionaries to create a new one.

Program

```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
dic4 = {}
print("Dictionary 1:",dic1)
print("Dictionary 2:",dic2)
```

```
print("Dictionary 3:",dic3)
print("Concatenated Dictionary:",end=' ')
for d in (dic1, dic2, dic3): dic4.update(d)
print(dic4)
```

Output

```
Dictionary 1: {1: 10, 2: 20}
Dictionary 2: {3: 30, 4: 40}
Dictionary 3: {5: 50, 6: 60}
Concatenated Dictionary: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6:
60}
```

37. Write a Python script to check if a given key already exists in a dictionary.

Program

```
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
x=int(input("Enter a key value:"))
if x in d:
    print('Key is present in the dictionary')
else:
    print('Key is not present in the dictionary')
```

Output

```
Enter a key value:2
Key is present in the dictionary
```

38. Write a Python script to generate and print a dictionary that contains number (between 1 and n) in the form (x, x*x).

Program

```
n=int(input("Enter a limit:"))
d = dict()
for x in range(1,n+1):
    d[x]=x*x
print(d)
```

Output

```
Enter a limit:4
{1: 1, 2: 4, 3: 9, 4: 16}
```

39. Write a Python script to print a dictionary where the keys are numbers between m and n (both included) and the values are square of keys.



Program

```
m=int(input("Enter a lower limit:"))
n=int(input("Enter an upper limit:"))
d={}
for x in range(m,n+1):
    d[x]=x**2
print(d)
```

Output

```
Enter a lower limit:2
Enter an upper limit:10
{2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

40. Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2000 (both included).

Program

```
nl=[]
for x in range(1500, 2000):
    if x%7==0 and x%5==0:
        nl.append(str(x))
print (','.join(nl))
```

Output

```
1505,1540,1575,1610,1645,1680,1715,1750,1785,1820,1855,1890,1925
,1960,1995
```

41. Write a Python program to convert temperatures to and from celsius, fahrenheit.

Program

```
temp = input("Input the  temperature you like to convert? (e.g.,
45F, 102C etc.) : ")
degree = int(temp[:-1])
i_convention = temp[-1]
if i_convention.upper() == "C":
  result = int(round((9 * degree) / 5 + 32))
  o_convention = "Fahrenheit"
elif i_convention.upper() == "F":
  result = int(round((degree - 32) * 5 / 9))
  o_convention = "Celsius"
else:
  print("Input proper convention.")
```

```
print("The temperature in", o_convention, "is", result,
"degrees.")
```

Output

```
Input the  temperature you like to convert? (e.g., 45F, 102C
etc.) : 37C
The temperature in Fahrenheit is 99 degrees.
```

42. Write a Python program to construct the following pattern, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Program

```
n=5
for i in range(n):
    for j in range(i):
        print ('* ', end="")
    print('')
for i in range(n,0,-1):
    for j in range(i):
        print('* ', end="")
    print('')
```

Output

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```



43. Write a Python program that accept a word from the user and reverse it.

Program

```
word = input("Input a word to reverse:")
for char in range(len(word)- 1, -1, -1):
  print(word[char], end="")
```

Output

```
Input a word to reverse:Python
nohtyP
```

44. Write a Python program that counts odd and even numbers from a list.

Program

```
lis=input("Enter some positive integers(space separated):")
numbers=list(map(int,lis.split()))
count_odd = 0
count_even = 0
for x in numbers:
        if not x % 2:
          count_even+=1
        else:
          count_odd+=1
print("Number of even numbers :",count_even)
print("Number of odd numbers :",count_odd)
```

Output

```
Enter some positive integers(space separated):1 2 3 4 5 6 7 8
Number of even numbers : 4
Number of odd numbers : 4
```

45. Write a Python program which takes two digits m (row) and n (column) as input and generates a two dimensional array. Read the elements and display the array.

Program

```
row_num = int(input("Input number of rows: "))
col_num = int(input("Input number of columns: "))
multi_list = [[0 for col in range(col_num)] for row in
range(row_num)]
for row in range(row_num):
    for col in range(col_num):
        multi_list[row][col]= int(input("Enter the value:"))
print(multi_list)
```

Output

```
Input number of rows: 2
Input number of columns: 2
Enter the value:1
Enter the value:2
Enter the value:3
Enter the value:4
[[1, 2], [3, 4]]
```

46. Write a Python program that accepts sequence of lines (blank line to terminate) as input and prints the lines as output (all characters in lower case).

Program

```
lines = []
while True:
    l = input("Enter a line:")
    if l:
        lines.append(l.lower())
    else:
        break
for l in lines:
    print(l)
```

Output

```
Enter a line:python
Enter a line:Programming
Enter a line:Is
Enter a line:Fun
Enter a line:
python
programming
is
fun
```

47. Write a Python program which accepts a sequence of comma separated 4 digit binary numbers as its input and print the numbers that are divisible by 5 in a comma separated sequence.

Program

```
items = []
num=input("Enter some binary numbers(comma separated):")
numl=list(num.split(','))
```



```
for p in numl:
    x = int(p,2)
    if not x%5:
        items.append(p)
print(','.join(items))
```

Output

```
Enter some binary numbers(comma separated):0101,1100,1111,1010
0101,1111,1010
```

48. Write a Python program that accepts a string and calculate the number of digits, letters and other characters.

Program

```
s = input("Input a string:")
d=l=a=0
for c in s:
    if c.isdigit():
        d=d+1
    elif c.isalpha():
        l=l+1
    else:
        a+=1
print("Letters:", l)
print("Digits:", d)
print("Other Characters:",a)
```

Output

```
Input a string:**python Programming123#
Letters: 17
Digits: 3
Other Characters: 4
```

49. Write a Python program to find numbers between 100 and 400 (both included) where each digit of a number is an even number. The numbers obtained should be printed in a comma-separated sequence.

Program

```
items = []
for i in range(100, 401):
  s = str(i)
```

```
    if (int(s[0])%2==0) and (int(s[1])%2==0) and (int(s[2])%2==0):
        items.append(s)
print(",".join(items))
```

Output

```
200,202,204,206,208,220,222,224,226,228,240,242,244,246,248,260,
262,264,266,268,280,282,284,286,288,400
```

## 5.7 Conclusion

This Chapter explains various types of decision making in Python which includes the `if, if ...else, if...elif...else`, nested loops with the flow charts and illustrated examples. Python's looping constructs which includes `for loop, for loop with else, while loop and while loop with else` are explained in detail. The two looping constructs `for loop with else` and `while loop with else` are absent in other programming languages like C, C++, Java etc. Similarly Python's control statements `break, continue` and `pass` are explained with illustrated examples. In this `pass` is a new type of control statement available only in Python. Various types of loops like infinite loops, loops with condition at the top, loops with condition in the middle and loops with condition at the bottom is also covered in this Chapter. This Chapter has provided solved programs which aid a user in Python programming.

## 5.8 Review Questions

1. Explain various decision making statements in Python.
2. Explain for loop with else with an example.
3. What is the purpose of break statement?
4. Differentiate while loop and while loop with else with examples.
5. What is the purpose of continue statement?
6. What is the purpose of pass statement in Python?
7. Differentiate break and continue statements.
8. Explain different types of loops with examples.



# 06 Functions

CONTENTS



A group of related statements to perform a specific task is known as a function. Functions help to break the program into smaller units. Functions avoid repetition and enhance code reusability. Functions provide better modularity in programming. Python provides two types of functions.

a) Built-in functions
b) User-defined functions

Functions like input(), print() etc. are examples of built-in functions. The code of these functions will be already defined. We can create our own functions to perform a particular task. These functions are called user-defined functions.

## 6.1 Function Definition

The following specifies simple rules for defining a function.

- Function block or Function header begins with the keyword def followed by the function name and parentheses ( ( ) ).
- Any input parameters or arguments should be placed within these parentheses. We can also define parameters inside these parentheses.