Python Programming with Applications: From Basics to Advance

PROF. AMIT KUMAR MISHRA
DR. DIPAK PANDURANG PATIL
DR. TUSHAR H JAWARE

Xoffencer

Python Programming with Applications: from Basics to Advance

Authors:

- Prof Amit Kumar Mishra
- Dr. Dipak Pandurang Patil
- Dr. Tushar H. Jaware

Xoffencer

www.xoffencerpublication.in

ISBN-13: 978-81-964018-7-0 (paperback)

Publication Date: 4 July 2023

MRP: ₹450/-

Published by:

Xoffencer International Publication

Behind shyam vihar vatika, laxmi colony

Dabra, Gwalior, M.P. – 475110

Cover Page Designed by:

Satyam soni

Contact us:

Email: mr.xoffencer@gmail.com

Visit us: www.xofferncerpublication.in

Author Details

Prof Amit Kumar Mishra

Prof Amit Kumar Mishra pursued his Bachelor of Engineering from Shantilal Shah Engineering College (Government College) Bhavnagar Gujarat in Electronics and Communication branch & did his M.E in Communication Engineering from MIT Aurangabad Maharashtra. He is pursuing PhD from department of Science and Technology of Savitribai Phule Pune University Pune in Electronics & Telecommunication domain. Presently he is working as Assistant Professor in E & TC Department of Sandip Institute of Engineering and Management Nashik Maharashtra India. He has total 14 years of teaching experience. He received grant of Rupees One Lakh from BCUD Savtribai Phule Pune University Pune. He has total 20 publications in various International & National Conferences and Journals. He has membership of different professional bodies like ISTE, IEI, INAAR, I2OR. Also, Prof Amit Mishra has received three Awards (one International and two National), he has published four books and two chapters. One of his chapter named “Health Detection System for COVID-19 Patients Using IoT” (Book Title - Medical Imaging and Health Informatics) is published in Scopus. Prof Amit Mishra got felicitated by Hon’ble Education minister of Maharastra Mr Chandrakant Patil during an event “Engineering Talent Search 2022” for being Jury member of the event.

Dr. Dipak Pandurang Patil

Dr. Dipak Pandurang Patil, currently working as a Professor and Principal at Sandip Institute of Engineering and Management Nashik Maharashtra India. In addition, he is Dean of International Affairs. He has more than 20 years of experience in teaching. He has completed PhD in Electronics and Telecommunication Engineering from Sant Gadge Baba Amravati University in Electronics & Telecommunication Engineering. He has more than 30 publications to his credit in reputed International Journals and Conferences along with five book chapters and three Indian patent on his research. He has membership of different National & International professional bodies like IEEE, ISTE, EAI and IAENG. He has delivered expert talks and sessions in various Institutes and conferences, and served as session chair and designated reviewer in reputed international conferences. In addition to this, he has also received awards at state and national level. He has participated in ERASMUS+ mobility project and received prestigious YOUTHPASS of European Commission.

Dr. Tushar H. Jaware

Dr. Tushar H. Jaware holds a bachelor's degree in electronics and telecommunication engineering from North Maharashtra University, Jalgaon. He further pursued a master's degree in digital electronics and obtained a Ph.D. in medical image processing from Sant Gadge Baba Amravati University, Amravati. Currently serving as the Dean of Research and Development at the R. C. Patel Institute of Technology in Shirpur, Maharashtra, India, Dr. Jaware possesses over 18 years of invaluable teaching experience. He is widely recognized as a Ph.D. Supervisor in electronics engineering at North Maharashtra University, Jalgaon, and Dr. Babasaheb Ambedkar Technological University, Lonere. Furthermore, he has contributed as a Member of the Board of Studies in electronics and telecommunication engineering at North Maharashtra University, Jalgaon. Demonstrating his innovative prowess, Dr. Jaware holds three international and national patents, along with six copyrighted works. His research findings have been published in 62 esteemed research papers featured in renowned international journals and conferences. His expertise in the field has garnered invitations as a Plenary Speaker to numerous prestigious events. Dr. Jaware has been bestowed with several accolades, including the Loksatta Tarun Tejankit Award in 2019 and the GIS Young Innovator and Researchers Award (Central India) in 2018, presented by JSR Laboratory, Pune, in collaboration with the Asian Society for Scientific Research. He also received the esteemed 'Bright Researcher Award' from the International Institute of Organized Research in 2017. Additionally, Dr. Jaware has been honored with 12 awards recognizing his outstanding research and academic contributions by various societies. Notably, he has secured research grants from AICTE under the SPICES scheme and under the Unnat Bharat Abhiyaan initiative.

Preface

Welcome to "Python Programming: From Basics to Advanced." This book is your gateway to the dynamic and captivating world of Python, where you'll discover the incredible power and versatility of this popular programming language.

Python's rise to prominence in the programming world is no coincidence. It's an elegant language that strikes the perfect balance between simplicity and functionality. Whether you're a programming novice or an experienced developer, Python welcomes you with its clean syntax and ease of use. Its intuitive nature makes it an ideal language to learn, and its powerful libraries and frameworks enable you to tackle an array of applications, from web development to data science and artificial intelligence.

Our mission is to equip you with the skills needed to confidently navigate the Python landscape, regardless of your experience level. We'll begin with the fundamentals, taking you through variables, data types, and control structures. Gradually, we'll dive deeper into functions, object-oriented programming, and advanced concepts like decorators and generators.

As you progress, you'll embark on exciting journeys into real-world Python applications.

Whether you're a curious beginner or a seasoned programmer seeking to expand your skillset, this book is designed with you in mind. For newcomers, we provide a gentle introduction to programming concepts, while experienced developers will appreciate the comprehensive coverage of Python's advanced features and applications.

Each chapter is carefully crafted to offer standalone value, enabling you to jump into specific topics or follow the logical progression from beginning to end. Practical examples and exercises are sprinkled throughout to reinforce your learning and empower you to experiment with Python code.

As authors, we take pride in presenting you with original, well-researched content that is free from plagiarism. We've put in the effort to ensure that the knowledge

shared here is both accurate and unique. Any external sources used are appropriately cited, giving you confidence in the authenticity of this book.

Creating this book wouldn't have been possible without the support and encouragement of the Python community. We extend our gratitude to the countless developers, contributors, and enthusiasts who have helped shape Python into the exceptional language it is today.

It's time to seize the opportunity and embrace Python's potential. Whether you aspire to build web applications, delve into data analytics, or explore the realms of artificial intelligence, Python is the perfect companion for your programming adventures. So, grab your keyboard, fire up your enthusiasm, and let's embark on this incredible journey together!

Happy coding!

Prof Amit Mishra
Dr Dipak P Patil
Dr Tushar H Jaware

4 Aug 2023

Contents

INTRODUCTION

- Python is a powerful, high level programming language.
- Python is a scripting language that is interpreted.
- Python programming is credited to Guido Van Rossum as its creator.
- Python is a dynamic, high-level, general-purpose, and interpreted programming language. It offers a large number of high-level data structures and is straightforward and simple to learn.
- Python is a programming language that is appealing for application development since it is simple to learn yet also strong and flexible.
- Since the variables are dynamically typed, we can simply write a=10 to assign an integer value to an integer variable without using data types to specify them..

Python History and Versions

- Late in the 1980s, Python began to take shape.
- Guido Van Rossum at CWI in the Netherlands began implementing Python in December 1989.
- It was first made available on February 20, 1991.

Python Features

The following list of features offered by Python:

1) **Simple to Use and Learn**

   Python is simple to use and learn. It is a high-level programming language that is user-friendly to developers.

2) Expression of Ideas

Because the Python language is more expressive, it is also easier to read and understand.

3) Interpretation

Python is an interpreted language, meaning that the code is run line by line. Debugging is now simple and ideal for novices.

4) Use of Free and Open Source

The Python programming language is available for free at www.python.org. There is also access to the source code. It is therefore open source.

5) Big Standard Library

Python is a sizable and diverse library and offers a vast collection of modules and functions for the quick construction of applications.

6) Support for GUI Programming

Graphical user interfaces (GUIs)Here, we are specifying applications areas where python can be applied:

1) Web Applications
2) Desktop GUI Applications
3) Software Development
4) Scientific and Numeric

Python is frequently used in scientific and numerical computation and is quite well known. SciPy, Pandas, IPython, and other helpful libraries and

packages are available. SciPy is a collection of math, scientific, and engineering software packages.

5) Business Software
   Business applications like ERP and e-commerce platforms are created using Python.

Various IDE for Python

There are various IDE available for implementing Python. Few of them are as follows:

- PYCHARM
- Thonny
- Anaconda
- IDLE (inbuild while installing python)

Any of the above tool can be used for executing the codes of Python.

TASKSHEET – 1

Python Installation

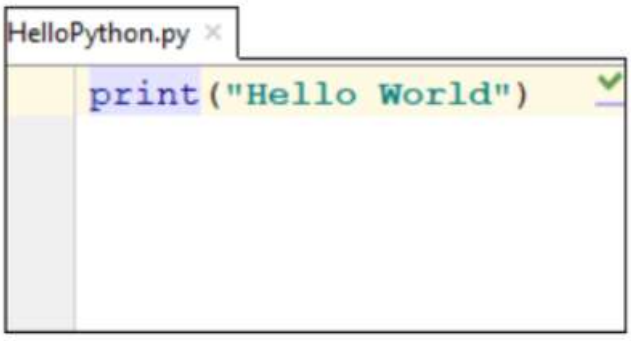
Install Python

From a Command Prompt or Terminal window, you can now type and execute your code. But that is quite tedious. We're going to use "Thonny", a piece of software.

Versions are available for Windows, Linux, and Mac.

You can find the software download page here:

First Python Program



OR

INSTALL ANACONDA IDE

TASKSHEET – 2

Python Variables

Creating Variables

Python has no command for declaring variables, in contrast to other programming languages. A variable is created once a value is assigned to it.

Example

```
x = 5
y = "Sumit"
print(x)
print(y)
```

Variables can change types after they have been set and are not required to be defined with a certain type.

Example

```
x = 4 # x is of type int
x = "Priya" # x is now of type str
print(x)
```

Variable Names

A variable's name can be short (like x and y) or long (like age, name, or total_volume).

Python variable rules:

- A variable name must begin with a letter or an underscore;

- it cannot begin with a number;
- it can only contain alphanumeric letters and underscores (A-z, 0-9, and _);
- Case-sensitivity applies to variable names (age, Age, and AGE are all distinct variables).

VARIABLES ARE CASE-SENSITIVE, SO KEEP THAT IN MIND.

Output Variables

The Python **print** statement is often used to output variables.

Python uses the + symbol to join text and a variable::

Example

```
x = "interesting"
print("Python is " + x)
```

You can also use the + character to add a variable to another variable:

Example

```
x = "Python is "
y = "interesting"
z = x + y
print(z)
```

For numbers, the + character works as a mathematical operator:

Example

```
x = 50
y = 10
print(x + y)
```

Python will give you the error if you attempt to combine a string and a number:

Example

```
x = 50
y = "Sumit"
print(x + y)
```

TASKSHEET – 3

Python Numbers

In Python, there are three types of numbers:

- Int
- Float
- complex

When you give a variable of a numeric type a value, you create a numeric type variable:

Example

```
x = 1 # int
y = 2.8 # float
z = 1j # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

Int

A number, positive or negative, without decimals, and with an unlimited length is known as a "integer."

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

A positive or negative number with one or more decimals is referred to as a "float," or "floating point number."

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Complex

Complex numbers are written with as "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Python Casting

Specify a Variable Type

You might occasionally want to assign a type to a variable. Casting can be used for this. Python uses classes to describe data types, including its primitive kinds, as it is an object-oriented language.

Consequently, constructor functions are used for casting in Python:

- **int() -** creates an integer number from a string literal that represents a whole number, a float literal that rounds down to the previous whole number, or an integer literal.
- **float() –** creates a float number from an integer, float, or string literal, provided the string is an integer or float.
- **str() –** creates a string from a number of data types, such as strings, integer literals, and float literals.

Example

Integers:

```
x = int(1) # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Example

Floats:

```
x = float(1) # x will be 1.0
y = float(2.8) # y will be 2.8
z = float("3") # z will be 3.0
w = float("4.2") # w will be 4.2
```

Example

Strings:

```
x = str("s1") # x will be 's1'
y = str(2) # y will be '2'
z = str(3.0) # z will be '3.0'
```

TASKSHEET – 4

Python Strings

String Literals

In Python, single or double quotation marks should be used to delimit string literals.

'hello' is the same as "hello".

The print function can be used to output strings to the screen. like this: print("hello").

Python's strings, like those of many other widely used programming languages, are collections of bytes that represent unicode characters. Python does not, however, support character data types; instead, a single character is represented as a string with length 1.

To access the string's , use square brackets.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Techpheonix!"
print(a[1])
```

Example

Substring. Get the characters from position 3 to position 7 (not included):

```
b = " Techpheonix!"
print(b[3:7])
```

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = "  Techpheonix "
print(a.strip()) # returns "Techpheonix"
```

Example

The len() method returns the length of a string:

```
a = "Techpheonix"
print(len(a))
```

Example

The lower() method returns the string in lower case:

```
a = "Techpheonix"
print(a.lower())
```

Example

The upper() method returns the string in upper case:

```
a = "Techpheonix"
print(a.upper())
```

Example

The replace() method replaces a string with another string:

```
a = "Newsoft Computers"
print(a.replace("N", "J"))
```

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Newsoft, Computers"
print(a.split(",")) # returns ['Newsoft', ' Computers']
```

TASKSHEET – 5

Python Operators

Python Operators

Operations on variables and values are carried out using operators.

The operators in Python are split into the following categories:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

**	Exponentiation	x ** y
//	Floor division	x // y

```
x = 3
y = 2

# Output: x + y = 5
print('x + y =',x+y)

# Output: x - y = 1
print('x - y =',x-y)

# Output: x * y = 6
print('x * y =',x*y)

# Output: x / y = 1.5
print('x / y =',x/y)

# Output: x // y = 1
print('x // y =',x//y)

# Output: x ** y = 9
print('x ** y =',x**y)
```

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

x = 10

y = 12

 # Output: x > y is False

print('x > y is',x>y)

 # Output: x < y is True

print('x < y is',x<y)

 # Output: x == y is False

print('x == y is',x==y)

 # Output: x != y is True

print('x != y is',x!=y)

 # Output: x >= y is False

print('x >= y is',x>=y)

 # Output: x <= y is True

print('x <= y is',x<=y)

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

x = True

y = False

 # Output: x and y is False

print('x and y is',x and y)

 # Output: x or y is True

print('x or y is',x or y)

 # Output: not x is False

print('not x is',not x)

Python Identity Operators

Identity operators are used to compare objects to determine whether they are indeed the same object in the same memory address rather than whether they are equal:

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
```

 # Output: False

```
print(x1 is not y1)
```

 # Output: True

```
print(x2 is y2)
```

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
x = 'Hello world'
# Output: True
print('H' in x)
```

 # Output: False

```
print('Hello' not in x)
```

TASKSHEET – 6

Python Lists

There are four collection data types in the python programming language:

- **List** is a collection which is ordered and changeable, indexed. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Understanding a collection type's characteristics is helpful when selecting one. The appropriate type selection for a given data set may result in the retention of meaning as well as an improvement in efficiency or security.

List

A list is a collection which is ordered and changeable. In python lists are written with **square brackets.**

Example

Create a list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Access Items

You access the list items by referring to the index number:

Example

print the second item of the list:

```
thislist = ["Apple", "Banana", "Cherry"]
print(thislist[1])
```

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = [“apple”, “banana”, “cherry”]
thislist[1] = “mango”
print(thislist)
```

Loop through a list

You can loop through the list items by using a for loop:

Example

print all items in the list, one by one:

```
thislist = [“apple”, “banana”, “cherry”]
for x in thislist:
    print(x)
```

You will learn more about for loops in out Python For loop chapters.

Check if Item Exists

To determine if a specified item is present in a list use in the keyword:

Check if item Exists

To determine if a specified item is present in a list use in the keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
    print("Yes , 'apple' is in the fruits list")
```

List Length

To determine how many item a list has, use the len() method:

Example

print the number of items in the list:

```
thislist = ["apple", "banana","cherry"]
print(len(thislist))
```

Add Items

To add an item to the end of the list, use the **append()** method:

Example

Using the **append()** method to append an item:

```
thislist = [“apple’, “banana”,”cherry”]
thislist.append(“orange”)
print(thislist)
```

To add an item at the specified index, use the insert() method:

Example

Insert n item as the second position:

```
thislist= [“apple”, “banana”, “cherry”]
thislist.insert(1, “tomoto”)
print(thislist)
```

Remove Item

There are several methods to remove items from a list:

Example

The remove() method removes the specified item:

```
thislist = [“apple”,”banana”,”cherry”]
thislist.remove(“banana”)
print(thislist)
```

Example

The pop() method removes the specified index , (or the last item if index is not specified):

```
thislist = [“apple”, “banana”, ”cherry”]
thislist.pop()
print(thislist)
```

Example

The **del** keyword removes the specified index:

```
thislist = [“apple”, “banana”, ”cherry”]
del thislist[0]
print(thislist)
```

Example

The **del** keyword removes the specified index:

```
thislist = [“apple”,”banana”,”cherry”]
del thislist[0]
print(thislist)
```

Example

The del keyword can also delete the list completely:

```
thislist = [“apple”, “banana”, “cherry”]
del thislist
```

Example

The clear() method empties the list:

```
thislist = [“apple”, “banana”, “cherry”]
thislist.clear()
```

```
print(thislist)
```

Copy a List

You cannot copy a list simply by typing list2 = list1, because : list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in list Method copy().

Example

Make a copy of a list with the list() method:

```
thislist = [“apple”, “banana”, “cherry”]
mylist = list(thislist)
print(mylist)
```

The list() Constructor

It is also possible to use the list() constructor to make a new list.

Example

Using the list() constructor to make a list:

```
thislist = list((“apple”, “banana”, “cherry”))
#note the double round brackets
print(thislist)
```

Count()

The count() method returns the number of elements with the specified value.

Example

```
mylist = [10,20,30,50,70,80,90,30,50]
X = mylist.count(50)
print(X)
```

Extend()

The **extend()** method adds the specified list elements (or any iterable) to the end of the current list.

Example

```
thislist = [“apple”, “banana”, “cherry”]
yourlist = [“ginger”, “turmeric”, “carrot”]
thislist.extend(yourlist)
print(thislist)
```

Index()

The index() method returns the position at the first occurrence of the specified value.

Example

```
x= thislist.index(“ginger”)
print(x)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

TASKSHEET – 7

Python Tuples

Tuple

A tuple is index, ordered collection that cannot be changed. Tuples are written in round brackets in Python.

Example

Create a Tuple:

```
thistuple = (“apple”, “banana”, “cherry”)
print(thistuple)
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Return the item in position 1:

```
thistuple = (“apple”, “banana”, “cherry”)
print(thistuple[1])
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable.

Loop Through a Tuple

You can loop through the tuple item by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = (“apple”, “banana”, “cherry”)
for x in thistuple:
    print(x)
```

You will learn more about for loops in out Python For loop chapters.

Check if Items Exists

To determine if a specified item is present in a tuple use the **in** keyword:

Example

Check if “apple” is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Yes, ‘apple’ is in the fruits tuple")
```

Tuple Length

To determine how many items a tuple has , use the len() method:

Example

print the number of items in the tuple:

```
thistuple = (“apple”, “banana”, “cherry”)
print(len(thistuple))
```

Add Items

Once a tuple is created, you cannot add items to it. Tuples are unchangeable.

Example

You cannot add items to a tuple:

```
thistuple = (“apple”, “banana”, “cherry”)
thistuple[3] = “orange”
print(thistuple)
```

Remove Items

Note: You cannot remove items in a tuple.

Because tuples are immutable, you cannot remove any of its element, but you can totally remove the entire tuple.

Example

The del keyword can delete the tuple completely:

```
thistuple = (“apple”, “banana”, “cherry”)
del thistuple
print(thistuple)
```

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

```
thistuple = tuple((“apple”, “banana”, “cherry”))
print(thistuple)
```

Tuple Methods

Python has two built – in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

TASKSHEET – 8

Python Set

Set

A set is a collection which is **unordered** and **unindexed**. In python sets are written with curly brackets.

Example

Create a Set:

```
thisset = {“apple”, “banana”, “cherry”}
print(thisset)
```

Note: Sets are unordered, so the items will appear in a random order.

Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a **for** loop, or ask if a specified value is present in a set, by using the **in** keyword.

Example

Loop through the set, and print the values:

```
thisset = {“apple”, “banana”, “cherry”}
for x in thisset:
        print(x)
```

Example

Check if “banana” is present in the set:

```
thisset = {“apple”, “banana”, “cherry”}
print(“banana” in thisset)
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add items

To add one items to a set use the add() method.

To add more than one item to a set use the update() method.

Example

Add an item to a set using the add() method:

```
thisset = {“apple”, “banana”, “cherry”}
thisset.add(” orange”)
print(thisset)
```

Example

Add multiple items to a set, using the update() method.

```
thisset = {“apple”, “banana”, “cherry”}
thisset.update([“orange”, “mango”, “grapes”])
print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

```
thisset = {“apple”, “banana”, “cherry”}
print(len(thisset))
```

Remove Item

To remove an item in a set, use the **remove()**, or the **discard()** method.

Example

remove “banana” by using the remove() method:

```
thisset = {“apple”, “banana”, “cherry”}
thisset.remove(“banana”)
print(thisset)
```

Note: If the item to remove does not exist, remove () will raise an error.

Example

Remove “banana” by using the discard() method:

```
thisset = {“apple”, “banana”, “cherry”}
thisset.discard(“banana”)
print(thisset)
```

The return value of the pop() method is the removed item.

Example

Remove the last item by using the pop() method:

```
thisset = {“apple”, “banana”, “cherry”}
x=thisset.pop()
print(x)
print(thisset)
```

Example

The **clear()** method empties the set:

```
thisset = {“apple”, “banana”, “cherry”}
thisset.clear()
print(thisset)
```

Example

The del keyword will delete the set completely:

```
thisset = {“apple”, “banana”, “cherry”}
del thisset
print(thisset)
```

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(( “apple”, “banana”, “cherry”))
print((thisset))
```

difference() Method

The difference() method returns a set that contains the difference between two sets.

```
x= {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”, “apple”}
z = x.difference(y)
print(z)
```

difference_update() Method

The items that are present in both sets are eliminated using the difference_update() method.

The difference_update() method differs from the difference() method in that it updates the old set by removing the undesired elements, whereas the difference() method returns a new set with the unwanted things removed.

```
x= {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”, “apple”}
x.difference_update(y)
print(x)
```

Intersection() Method

The intersection() method returns a set that contains the similarity between two or more sets.

```
x= {“apple”, “banana”, “cherry”}
```

```
y = {“google”, “microsoft”, “apple”}
z = x.intersection(y)
print(z)
```

intersection_update() Method

The elements that are absent from both sets are eliminated using the intersection_update() method.

The intersection_update() function differs from the intersection() method in that it updates the old set by removing the undesired elements, whereas the intersection() method returns a new set with the unwanted things removed.

```
x = {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”, “apple”}
x.intersection_update(y)
print(x)
```

isdisjoint() method

The isdisjoint() method returns True if none of the items are present in both sets, otherwise it returns False.

```
x = {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”, “apple”}
z= x.isdisjoint(y)
print(z)
```

issubset() Method

The issubset() method returns True if all items in the specified set exists in the specified set, otherwise it returns False.

```
x = {“f”, “e”, “d”, “c”, “b”, “a”}
y ={“a”, “b” “c”}
z= x.issubset(y)
```

issuperset() Method

The issuperset() method returns True if all items in the set exists in the original set, otherwise it returns False.

```
x = {“f”, “e”, “d”, “c”, “b”, “a”}
y = {“a”, “b”, “c”}
z = x.issuperset(y)
print(z)
```

symmetric difference() Method

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets.

```
x = {“apple”, “banana”, “cherry”}
y= {“google”, “microsoft”, “apple”}
z=x.symmetric_difference(y)
print(z)
```

symmetric_difference_update() Method

The symmetric_difference_update() method updates the original set by removing items that are present in both sets, and inserting the other items.

```
x= {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”,”apple”}
```

```
x.symetric_difference_update(y)
print(x)
```

union() Method

All of the items from the original set as well as all of the required sets are included in the set that is returned by the union() method.

Sets can be specified in any number, separated by commas.

There will only be one instance of an item in the result if it appears in more than one set.

```
x= {“apple”, “banana”, “cherry”}
y = {“google”, “microsoft”,”apple”}
z = x.union(y)
print(z)
```

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set

discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)

TASKSHEET – 9

Python Dictionaries

A dictionary is an unordered, changeable, and indexed collection. Dictionary entries in Python are enclosed in curly brackets and contain both keys and values.

Example

Create and print a dictionary:

```
thisdict = {“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
print(thisdict)
```

Accessing Items

By using the key name for a dictionary, which is enclosed in square brackets, you can access its items:

Example

Get the value of the “model” key:

```
X= thisdict[“model”]
```

There is also a method called get() tat will give you the same result:

Example

Get the value of the “model” key:

X = thisdict.get(“model”)

Change Values

We can change the value of a specific item by referring to its key name:

Example

Change the “year” to 2018:

```
thisdict = {
“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
thisdict[“year”] = 2018
```

Loop Through a Dictionary

You can loop through a dictionary by using a **for** loop.

Although there are ways to return the values as well, when looping through a dictionary, the return value is the dictionary's keys.

Example

print all key names in the dictionary, one by one:

```
for x in thisdict:
    print(x)
```

Example

print all values in the dictionary, one by one:

```
for x in thisdict:
    print(thisdict[x])
```

Example

You can also use the values() function to return values of a dictionary:

```
for x in thisdict.values():
    print(x)
```

****

Loop through both keys and values, by using the items() function:

```
for x,y in thisdict.items():
    print(x,y)
```

Check if key Exists

To determine if a specified key is present in a dictionary use the **in** keyword:

Example

Check if “model” is present in the dictionary:

```
thisdict ={
“brand” : “Ford” ,
“model” : “Mustang” ,
“year” : 1964
}
If “model” in thisdict:
    print(“Yes, ‘model’ is one of the keys in the thisdict dictionary”)
```

Dictionary Length

To determine how many items a dictionary has, use the **len()** method.

Example

print the number of items in the dictionary:

```
print(len(thisdict))
```

Adding Items

The process of adding something to the dictionary involves creating a new index key and giving it a value:

Example

```
thisdict = {
“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
thisdict[“color”] = “red”
print(thisdict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The pop() method removes the item with the specified key name:

```
thisdict = {
“brand”: “Ford”;
“model”: “Mustang”;
“year”: 1964
}
thisdict.pop(“model”)
print(thisdict)
```

Example

The most recent item put is removed using the popitem() method. (in versions before 3.7, a random item is removed instead):

```
thisdict = {
“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
thisdict.popitem()
print(thisdict)
```

Example

The **del** keyword removes the item with the specified key name:

Example

The **del** keyword removes the item with the specified key name:

```
thisdict = {
“brand”: “Ford”,
```

```
"model": "Mustang",
"year": 1964
}
del thisdict["model"]
print(thisdict)
```

Example

The **del** keyword can also delete the dictionary completely:

```
thisdict = {
"brand": "Ford";
"model": "Mustang";
"year": 1964
}
del thisdict
print(thisdict)
#this will cause an error because "thisdict" no longer exists.
```

Example

The **clear()** keyword empties the dictionary:

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
thisdict.clear()
print(thisdict)
```

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1.

Because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, and way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the copy() method:

```
thisdict= {
“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
mydict = thisdict.copy()
print(mydict)
```

Another way to make a copy is to use the built-in method **dict().**

Example

Make a copy of a dictionary with dict() method.

```
thisdict = {
“brand”: “Ford”;
“model”: “Mustang”;
“year”: 1964
}
```

```
mydict = dict(thisdict)
print(mydict)
```

The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

Example

```
thisdict = dict(brand= “Ford”, model=“Mustang”, year= 1964)
print(thisdict)
```

setdefault() Method

The setdefault() method returns the value of the item with the specified key.

If the key does not exist, insert the key, with the specified value, see example below

Example 1

```
car = {
“brand”: “Ford”,
“model”: “Mustang”,
“year”: 1964
}
x= car.setdefault(“model”, “Bronco”)
print(x)
```

Example 2

```
car = {
“brand”: “Ford”;
```

“model”: “Mustang”;

“year”: 1964

}

x= car.setdefault(“color”, “white”)

print(x)

Dictionary Methods

You can use a variety of built-in methods on dictionaries in Python.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

TASKSHEET – 10

Python If…Else

Python Conditions and If statements

Python supports the standard mathematical logical conditions:

- Equals: a == b
- Not Equals: a !=b
- Less than: a < b
- Less than or equal to: a <=b
- Greater than: a>b
- Greater than or equal to: a >=b

There are many methods to employ these conditions, but **"if statements"** and loops seem to be the most popular.

The if keyword is used to create a **"if statement".**

Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, ad so we print to the screen "b is greater than a".

Indentation

Python relies on indentation, using whitespace, to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b =200
if b > a:
print("b is greater than a")  # you will get an error
```

elif

The **elif** keyword is pythons way of saying "if previous conditions were not true, then try this condition".

Example

```
a = 33
b =33
if b >a:
        print("b is greater than a")
elif a ==b:
        print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a =200
b = 33
if b >a:
        print("b is greater than a")
elif a ==b:
        print("a and b are equal")
else:
        print(" a is greater than b")
```

In this example a is greater to b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b". You can also have an else without the elif:

Example

```
a= 200
b =33
if b > a:
        print("b is greater than a")
else:
        print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print ("a is greater than b")
```

Short Hand If…Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
print("A")  if a > b else print("B")
```

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

```
print("A") if a >b else print("=") if a==b else print("B")
```

And

The **and** keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
if a > b and c > a:
```

```
        print ("Both conditions are True")
```

Or

The **or** keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
if a >b or a >c:
        print ("At least one of the conditions is True")
```

Program

```
#Largest of two numbers
    a = float(input("Enter first number:"))
    b = float(input("Enter second number:"))
    if b >a:
      print("b is greater than a")
    elif a ==b:
      print("a and b are equal")
    else:
     print("a is greater than b")

#Python program to find the largest number among the three input numbers

#take three numbers from user

    num1 = float(input("Enter first number:"))
    num2 = float(input("Enter second number:"))
    num3 = float(input("Enter three number:"))
```

```
if (num1> num2) and (num1>num3):
    largest = num1
elif (num2>num1) and (num2>num3):
   largest = num2
else:
  largest = num3
print("The largest number is", largest)
```

TASKSHEET – 11

Python Loops

There are two primitive loop commands in Python:

- While loops
- For loops

“while loop”

While a condition is true, a set of statements can be carried out using the while loop.

Example

print i as long as i is less than 6:

```
i =1
while i < 6:
  print(i)
  i + = 1
```

Keep in mind that if you forget to increase i, the loop will never end.

In this example, we need to define an indexing variable, i, and set it to 1 in order to provide the necessary variables for the while loop.

The break Statement

Even though the while condition is true, the loop can be stopped with the break statement:

Example

Exit the loop when i is 3:

```
i = 1
While i < 6:
   print(i)
   if i == 3:
      break
   i +=1
```

With the continue statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i <6:
  i += 1
  if i ==3:
     continue
  print(i)
```

Python <u>for</u> loops

When iterating through a sequence (which can be a string, list, tuple, dictionary, set, or other object), a for loop is employed.

This functions more like an iterator method seen in other object-oriented programming languages and is less like the for keyword found in other programming languages.

With the help of the for loop, we may run a series of instructions once for each element of a list, tuple, set, etc.

Example

print each fruit in a fruit list:

```
fruits = [“apple”, “banana”, “cherry”]
for x in fruits:
    print(x)
```

The **for** loop does not require an indexing variable to set beforehand.

Looping through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word “banana”:

```
for x in “banana”:
        print(x)
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is “banana”:

```
fruits = [“apple”, “banana”, “cherry”]
for x in fruits:
```

```
print(x)
if x == “banana”:
break
```

Example

Exit the loop when x is “banana”, but this time the break comes before the print:

```
fruits = [“apple”, “banana”, “cherrry”]
for x in fruits:
  if x == “banana”:
    break
  print(x)
```

The Continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
fruits = [“apple”, “banana”, “cherrry”]
for x in fruits:
  if x == “banana”:
    continue
  print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default) , and ends at a specified number.

Example

Using the range() function:

```
for x in range(6):
    print(x)
```

Keep in mind that range(6) only includes the values 0 to 5, not 0 to 6.

The starting value for the range() function is 0 by default, but you may change it by adding a parameter: range(2,6), which means values between 2 and 6 (but not 6).

Example

Using the start parameter:

```
for x in range(2,6):
  print(x)
```

By default, the range() method increments the sequence by 1, but a third parameter can be added to indicate a different increment value:

```
range(2,30, 3):
```

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2,30,3):
   print(x)
```

else in for loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example

Print all the numbers from 0 to 5, then, when the loop is finished, print a message.

```
for x in range(6):
    print(x)
else:
  print(“Finally finished!”)
```

Nested Loops

A nested loop is a loop inside a loop.

Every time the "outer loop" iterates, the "inner loop" will be run once:

Example

print each adjective for every fruit:

```
adj = [“red”, “big”, “tasty”]
fruits = [“apple”, “banana”, “cherry”]
for x in adj:
  for y in fruits:
    print(x,y)
```

Programs :

#Program to print first ten numbers using while loop.

#Program to print first ten odd numbers using while loop.

#Program to print first ten even numbers using while loop.

#Program to print the table of 3 using while loop.

#Program to print first ten numbers using for loop.

#Program to print first ten even numbers using for loop.

#Program to print first ten odd numbers using for loop.

#Program to print all the even numbers between 1 to 789735.

#Program to print the first 989752 odd numbers.

TASKSHEET – 12

Python Functions

Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Creating a Function

In Python a function is defined using the **def** keyword:

Example

```
def my_function():
    print("Hello from a function")
my_function()
```

Parameters

Functions can accept information as a parameter. After the function name, parameters are listed between brackets. You can enter as many options as you like; simply comma-separate them.

The function (fname) in the following example only has one parameter. A first name is passed to the function when it is called, and it is utilised there to print the whole name:

Example

```
def my_function(fname):
        print(fname + "computers")
my_function("Techpheonix")
```

```
my_function("Newtech")
my_function("Sunrise")
```

Default Parameter Value

The following example shows how to use a default parameter value. If we call the function without parameter, it uses the default value:

Example

```
def my_function(country = "India"):
    print("I am from" + country)
my_function("sweden")
my_function("Japan")
my_function()
my_function("Brazil")
```

Return Values

To let a function return a value, use the **return** statement:

Example

```
def my_function(x):
  return 5*x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Programs

1. Program to print addition of two numbers

2. Program to print addition of three numbers
3. Program to print largest of two numbers
4. Program to print largest of three numbers

TASKSHEET – 13

Python Lambda

Python Lambda

Small anonymous functions are known as lambda.

A lambda function can have one expression but any number of arguments.

Syntax

Lambda arguments : expression

The expression is carried out, and the output is provided:

Example

A lambda function that prints the outcome after adding 10 to the number given as an argument:

```
x = lambda a: a +10
print(x(5))
```

Lambda functions can take any number of arguments:

Example

A lambda function that sums argument a,b and c and print the result:

```
x = lambda a,b,c : a + b + c
print(x(5,6,2))
```

Why use Lambda Functions?

When lambda is used as an anonymous function inside another function, their power is better demonstrated.

Let's say you have a function definition that takes a single parameter and multiplies that argument by an unknowable number:

```
def myfunc(n):
    return lambda a:a*n
```

Use that function definition to make a function that always double the number you send in:

Example

```
def myfunc(n):
    return lambda a : a *n
mydoubler = myfunc(2)
```

Or, use the same function definition to make a function that always triples the number you send in:

Example

```
def myfunc(n):
    return lambda a : a *n
mytripler = myfunc(3)
print(mytripler(11))
```

Or, use the same function definition to make both functions , in the same program

Example

```
def myfunc(n):
```

```
        return lambda a : a * n
mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(11))
print(mytripler(11))
```

When an anonymous function is needed for a brief length of time, use lambda functions.

TASKSHEET – 14

Python Classes – Objects

Python Classes and Objects

An object-oriented programming language is Python. In Python, almost everything is an object with properties and functions. A class functions as a kind of "blueprint" or object constructor.

Create a Class

Create a class named MyClass, with a property named x:

```
Class myClass:
    X = 5
```

Create Object

Now we can use the class named myClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = myClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in **__init__() function**. All classes have a function called **__init__()**, which is always executed when class is being initiated.

Example

Create a class named person, use the __init__() function to assign values for name and age:

```
class person:
  def __init__(self, name, age):
        self.name = name
        self.age = age
p1 = person ("John", 36)

print(p1.name)
print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods

Methods can also be found in objects. Object-specific functions are called methods in an object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
Class Person:
    def __init__ (self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is" + self.name)
p1 = Person ("John", 36)
p1.myfunc()
```

Note: The self-parameter, which is a reference to the active instance of the class, is used to access class-specific variables.

Delete Objects

You can delete objects by using the del keyword:

Example

```
Delete the p1 object:
  del p1
```

TASKSHEET – 15

Python Inheritance

By using inheritance, we can create a class that has all the methods and attributes of another class.

The class being inherited from, often known as the base class, is the parent class.

The class that inherits from another class is referred to as a child class or derived class.

Create a Parent Class

The syntax is the same as creating any other class because any class can be a parent class:

Example

Create a class named person, with firstname and lastname properties, and a printname method:

```
class person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:
    x = person("John", "Doe")
    x.printname()
```

Create a Child Class

Send the parent class as a parameter when constructing the child class to build a class that inherits the functionality from another class:

Example

Create a class called "student" that will take on the attributes and functions of the "person" class:

```
class student (person):
    pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Example

Use the Student class to create an object, and then execute the printname method:

```
x = Student("Mike", "Olsen")
x.printname()
```

Add the __init__() Function

So far we have created a child class that inherits the properties and methods from its parent. We want to add the __init__() function to the child class (instead of the pass keyword).

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Example

Add the __init__() function to the Student class:

```
class Student(Person):
        def __init__(self, fname, lname):
#add properties etc.
```

When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.

Note: The child's __init__() function overrides the inheritance of the parent's __init__() function.

To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:

Example

```
class Student(Person):
        def __init__(self, fname, lname):
                person.__init__(self, fname, lname)
```

Now we have successfully added the __init__() function, and kept the inheritance of the parent class, and we are ready to add functionality in the __init__() function.

Add Methods

Example

Add a method called welcome to the Student class:

```
class Student(Person):
```

```
    def __init__(self, fname, lname, year):
        person.__init__(self, fname, lname)
        self.graduationyear = year
    def welcome(self):
        print("Welcome", self.firstname, self.lastname, "to the class of",
        self.graduationyear)
```

The parent method's inheritance will be overridden if you introduce a method in the child class with the same name as a function in the parent class.

TASKSHEET – 16

Python Iterators

An object with a countable number of values is an iterator.

An object that can be iterated upon, or traversed through all the values, is known as an iterator.

Iterators are technically objects in Python that implement the iterator protocol, which consists of the methods __iter__() and __next__().

Iterator vs Iterable

Iterable objects include sets, dictionaries, lists, and tuples. You can obtain an iterator from them because they are iterable containers.

All these objects have a iter() method which is used to get an iterator:

Example

Return an iterator from a tuple, and print each value:

```
mytuple = (“apple”, “banana”, “cherry”)
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Even strings are iterable objects, and can return an iterator:

Example

Strings are also iterable objects, containing a sequence of characters:

```
mystr = “banana”
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Looping through an iterator

We can also use a for loop to iterate through an iterable object:

Example

Iterate the values of a tuple:

```
mytuple = (“apple”, “banana”, “cherry”)
for x in mytuple:
    print(x)
```

Example

Iterate the characters of a string:

```
mystr = “banana”
for x in mystr:
    print(x)
```

The for loop actually creates an iterator object and executes the next() method for each loop.

Create an Iterator

You must add the methods __iter__() and __next__() to your object in order to build an object or class that acts as an iterator.

All classes contain a function called __init__() that enables some initialising while the object is being formed, as you learnt in the Python classes and objects chapter.

Similar to this, the __iter__() method allows you to perform operations (initialising, etc.), but you must always return the iterator object.

You can perform actions using the __next__() method, which must return the subsequent element in the series.

Example

Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc):

```
class mynumbers:
        def __iter__(self):
                self.a = 1
                return self
        def __next__(self)
                x = self.a
                self.a +=1
                return x
myobj = mynumbers()
```

```
myiter = iter(myobj)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration

If you used enough next() instructions or a for loop, the above example would never end.

We can use the StopIteration statement to stop an iteration from continuing indefinitely.

We may add a terminating condition to the __next__() method to raise an error if the iteration is repeated a predetermined amount of times:

Example

stop after 20 iterations:

```
class mynumbers:
  def __iter__(self):
    self.a = 1
    return self
  def __next__(self):

    if self.a <=20:
      x = self.a
      self.a +=1
```

```
            return x
        else:
            raise StopIteration
myclass = mynumbers()
myiter = iter(myclass)

for x in myiter:
   print(x)
```

TASKSHEET – 17

Python Dates

The date time module in Python works with actual dates and times. The date and time must be used in real-world applications.

Python does not have a data type for dates, but we may import the datetime module to work with dates as date objects..

Example

Import the datetime module and display the current date:

```
import datetime
x = datetime.datetime.now()
print(x)
```

Date output

When we execute the code from the example above the result will be:

2021-07-04 17:22:30.670893

Year, month, day, hour, minute, second, and microsecond are all included in the data.

There are numerous ways to get data about a date object from the datetime module.

Here are a few illustrations; you may read more about these in the chapter's subsequent sections:

Example

Return the year and name of weekday:

```
Import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

The datetime() class (constructor) of the datetime module can be used to create dates.

Three parameters are needed to build a date using the datetime() class: year, month, and day.

Example

Create a data object:

```
import datetime
x = datetime.datetime(2020 , 5, 17)
print(x)
```

The hour, minute, second, microsecond, and tzone parameters are similarly taken by the datetime() class, but they are optional and have a default value of 0 (None for timezone).

The strftime() Method

Date objects can be formatted into readable strings using the datatime object's function.

The strftime() method only accepts one format parameter, which determines the format of the resulting string:

Example

Display the name of the month:

```
import datetime
x = datetime.datetime(2018,6,1)
print(x.strftime("%B"))
```

Python program to display weekday, short version

```
x = datetime.datetime.now()
print(x.strftime("%a"))
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18

%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM

import time

```
for i in range(0,5):
    print(i)
    #Each element will be printed after 1 second
    time.sleep(1)
import calender;
cal = calender.month(2020,9)
#printing the calendar of sept 2020
print(cal)
```

%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100

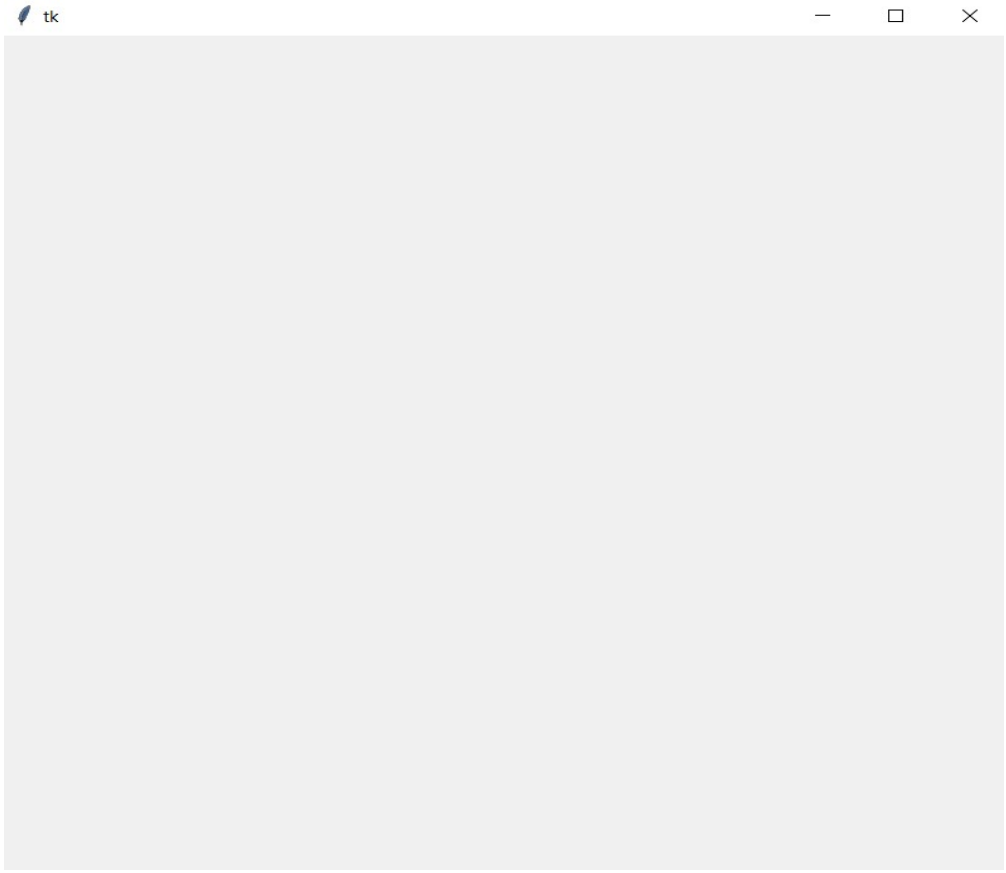
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

Python GUI

1. Program to Create Window

```
import tkinter
from tkinter import *
from tkinter.ttk import Combobox
window = tkinter.Tk()
window.geometry("800x800")
```

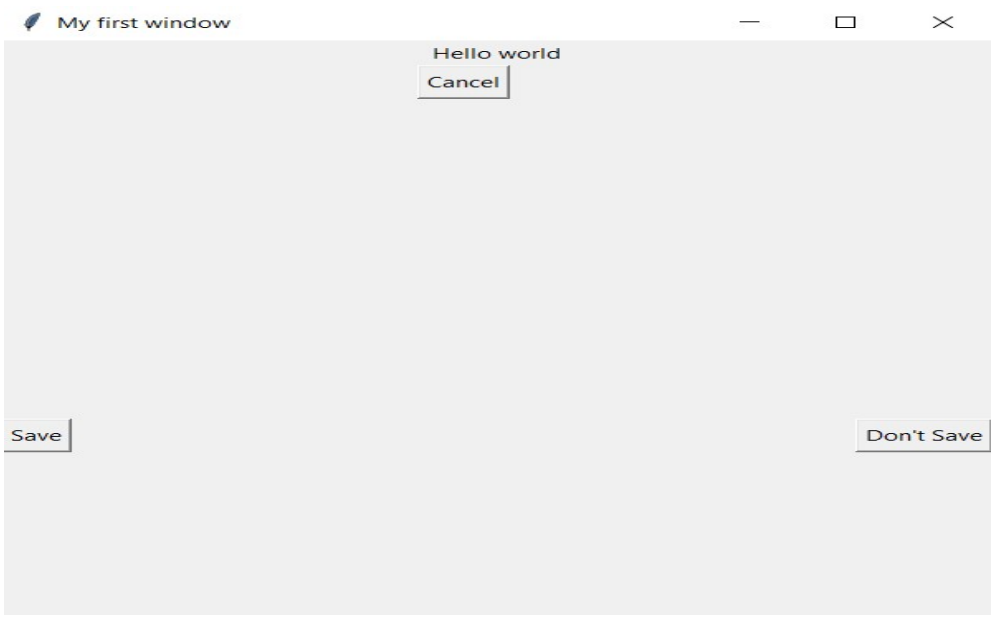
Output:



2) Program to Create Button

```
import tkinter
from tkinter import *
root=tkinter.Tk()
root.title("My first window")
root.geometry("600x800+400+400")
L1 = Label(root, text="Hello world")
L1.pack()
b1 = Button(root, text="Save")
b1.pack(side=LEFT)
b2 = Button(root, text="Don't Save")
b2.pack(side=RIGHT)
b3 = Button(root, text="Cancel")
b3.pack(side=TOP)
```

Output:



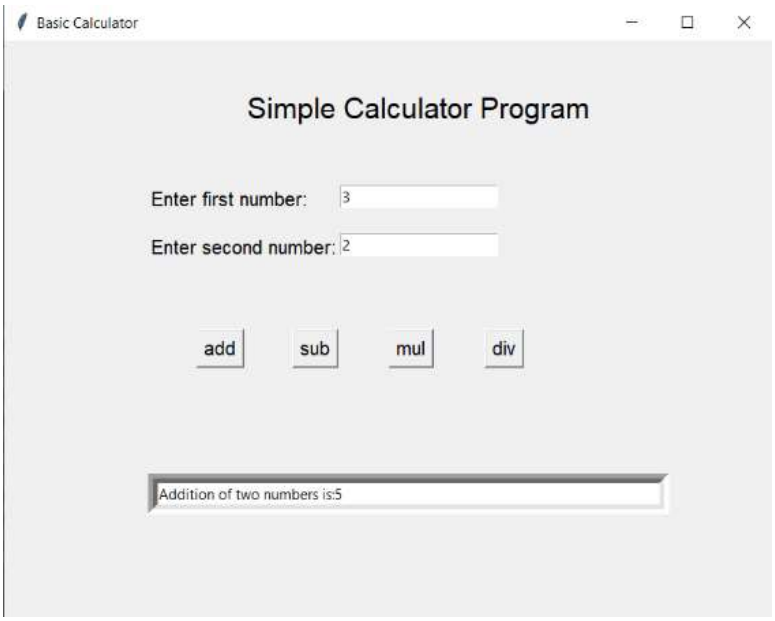
3) Program to make Calculator

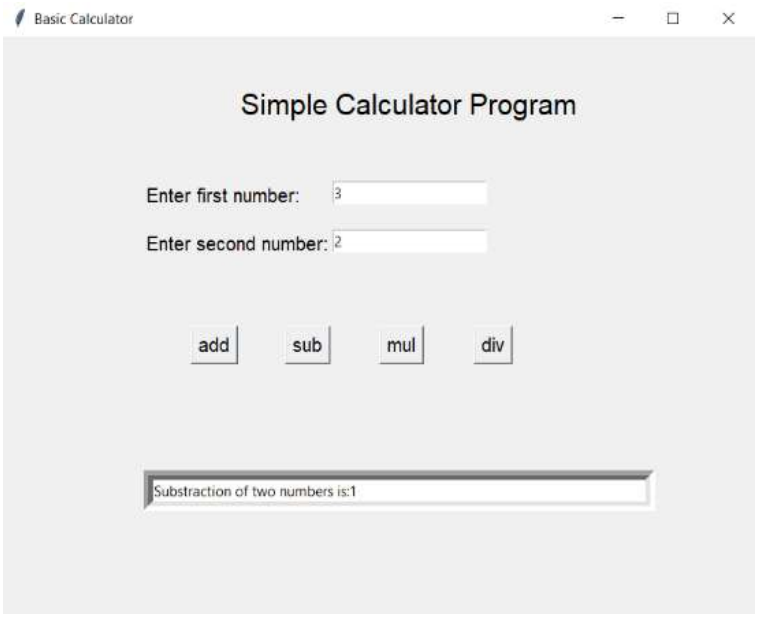
```
import tkinter
from tkinter import *

class calc:
   def sum(self):
      self.a = e1.get()
      self.b=e2.get()
      print("value of a is:", self.a)
      print("value of b is:", self.b)
      addition = int((self.a)) + int((self.b))
      print("Addition of the two numbers:", addition)
      result="Addition of two numbers is:"+str(addition)
      e3.delete(0,'end')
      e3.insert(END,result)

   def sub(self):

      self.a = e1.get()
      self.b=e2.get()
      print("value of a is:", self.a)
      print("value of b is:", self.b)
      substration = int((self.a)) - int((self.b))
      print("Substraction of the two numbers:", substration)
      result="Substraction of two numbers is:"+str(substration)
      e3.delete(0,'end')
      e3.insert(END,result)
```

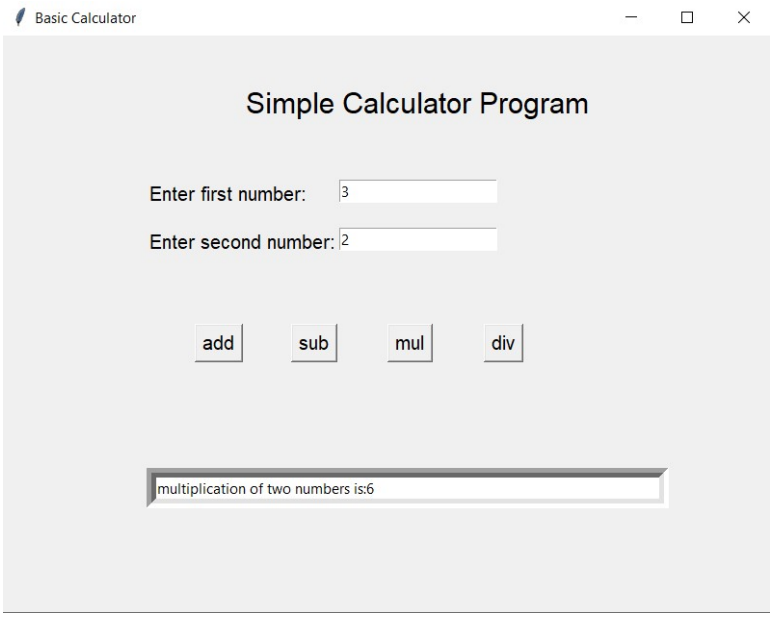
```
    def mul(self):

        self.a = e1.get()
        self.b=e2.get()
        print("value of a is:", self.a)
        print("value of b is:", self.b)
        multiplication = int((self.a)) * int((self.b))
        print("multiplication of the two numbers:", multiplication)
        result="multiplication of two numbers is:"+str(multiplication)
        e3.delete(0,'end')
        e3.insert(END,result)

    def div(self):

        self.a = e1.get()
        self.b=e2.get()
        print("value of a is:", self.a)
        print("value of b is:", self.b)
        division = int((self.a)) / int((self.b))
        print("division of the two numbers:", division)
        result="division of two numbers is:"+str(division)
        e3.delete(0,'end')
        e3.insert(END,result)


root = tkinter.Tk()
root.geometry("800x600")
root.title("Basic Calculator")
P = Label(root, text="Simple Calculator Program",
font=("Arial",18,"normal"))
```

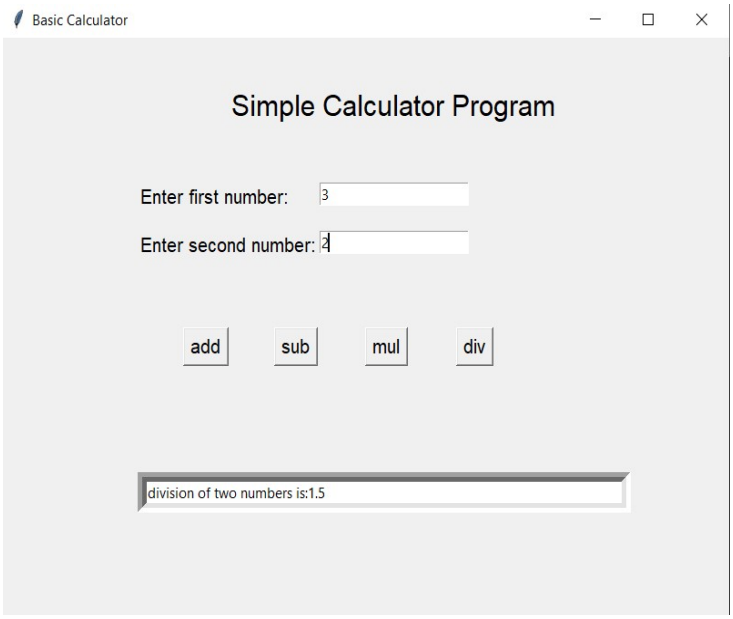
```
P.place(x=250, y=50)

Lab1=Label(root, text="Enter first number: ", font=("Arial",12,"normal"))
Lab1.place(x=150, y=150)
e1=Entry(root)
e1.place(x= 350, y=150)

Lab2=Label(root, text="Enter second number: ", font=("Arial",12,"normal"))
Lab2.place(x=150, y=200)
e2=Entry(root)
e2.place(x= 350, y=200)

x = calc()
add=Button(root,text="add",font=("Arial",12,"normal"),command=x.sum)
add.place(x=200, y= 300)

sub=Button(root,text="sub",font=("Arial",12,"normal"),command=x.sub)
sub.place(x=300, y= 300)

mul=Button(root,text="mul",font=("Arial",12,"normal"),command=x.mul)
mul.place(x=400, y= 300)

div=Button(root,text="div",font=("Arial",12,"normal"),command=x.div)
div.place(x=500, y= 300)

e3=Entry(root,bd=10, width=65)
e3.place(x= 150, y=450)
```

Output:


Basic Calculator
Simple Calculator Program
Enter first number:
3
Enter second number:
2
add
sub
mul
div
Addition of two numbers is:5


Basic Calculator
Simple Calculator Program
Enter first number:
3
Enter second number:
2
add
sub
mul
div
Substraction of two numbers is:1


Basic Calculator
Simple Calculator Program
Enter first number:
3
Enter second number:
2
add
sub
mul
div
multiplication of two numbers is:6


Basic Calculator
Simple Calculator Program
Enter first number:
3
Enter second number:
2
add
sub
mul
div
division of two numbers is:1.5

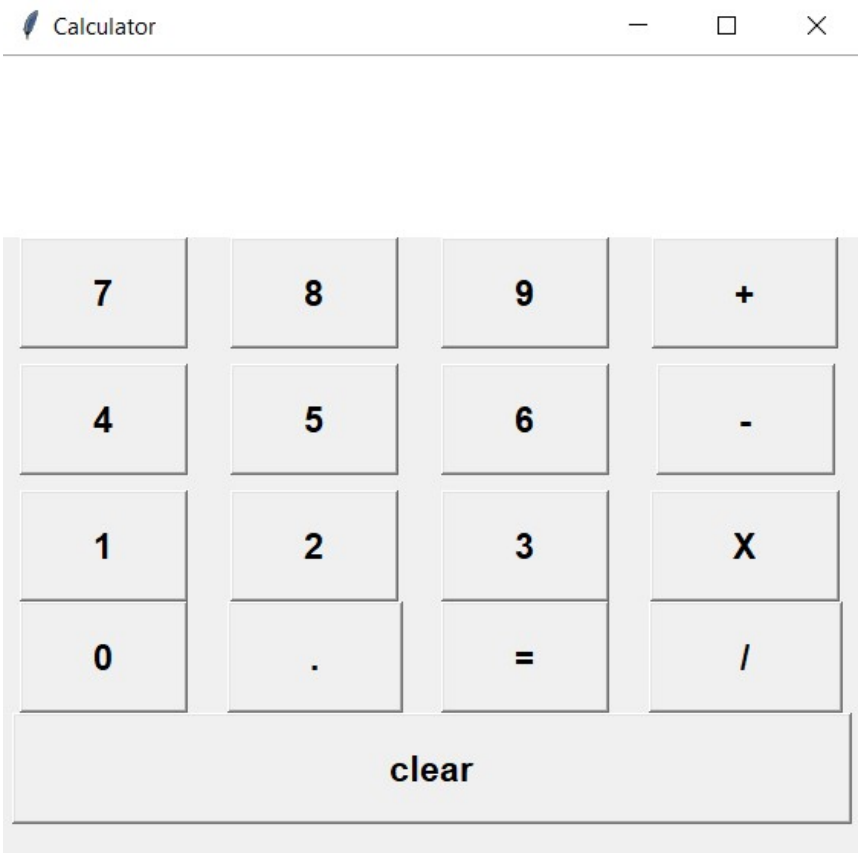
4) Program to create actual Calculator

```
import tkinter
import tkinter.font as font
from tkinter import *
from tkinter import font

root=tkinter.Tk()
root.title("Calculator")
root.geometry("564x630")
exp=""

def press(val):

    global exp
    exp=exp+val
    answer.set(exp)

def clear():

    global exp
    answer.set("")
    exp=""

def equal():

    total=eval(exp)
    answer.set(total)

answer=StringVar()
```

```
result=Entry(root, text=answer, font=("Arial Narrow", 20, "bold"),
justify="right")
result.grid(row=0, column=0, columnspan=4, ipadx=130, ipady=38)
fon=font.Font(family="Helvetica", size=14, weight="bold")
btn7=Button(root, text="7",font=fon, command=lambda:press("7"))
btn7.grid(row=1,column=0,ipadx=38, ipady=13)
btn8=Button(root, text="8",font=fon,command=lambda:press("8"))
btn8.grid(row=1,column=1,ipadx=38, ipady=13)
btn9=Button(root, text="9",font=fon,command=lambda:press("9"))
btn9.grid(row=1,column=2,ipadx=38, ipady=13)
btnplus=Button(root,text=" + ",font=fon,command=lambda:press("+"))
btnplus.grid(row=1, column=3, ipadx=38, ipady=13)
btn4=Button(root, text="4",font=fon,command=lambda:press("4"))
btn4.grid(row=2, column=0, ipadx=38, ipady=13, pady=10)
btn5=Button(root, text="5",font=fon, command=lambda:press("5"))
btn5.grid(row=2, column=1, ipadx=38, ipady=13)
btn6=Button(root, text="6",font=fon, command=lambda:press("6"))
btn6.grid(row=2, column=2, ipadx=38, ipady=13)
btnminus=Button(root, text=" - ",font=fon, command=lambda:press("-"))
btnminus.grid(row=2, column=3, ipadx=38, ipady=13)
btn1=Button(root, text="1",font=fon, command=lambda:press("1"))
btn1.grid(row=3, column=0, ipadx=38,ipady=13)
btn2=Button(root, text="2",font=fon, command=lambda:press("2"))
btn2.grid(row=3, column=1, ipadx=38,ipady=13)
btn3=Button(root, text="3",font=fon, command=lambda:press("3"))
btn3.grid(row=3, column=2, ipadx=38,ipady=13)
btnmultiply=Button(root, text=" X ",font=fon, command=lambda:press("*"))
btnmultiply.grid(row=3, column=3, ipadx=38, ipady=13)
```

```
btn0=Button(root, text="0",font=fon,command=lambda:press("0"))
btn0.grid(row=4, column=0, ipadx=38, ipady=13)
btndot=Button(root,text=" . ",font=fon,command=lambda:press("."))
btndot.grid(row=4, column=1, ipadx=38, ipady=13)
btnequal=Button(root,text= "=",font=fon,command=equal)
btnequal.grid(row=4, column=2, ipadx=38,ipady=13)
btndiv=Button(root,text="  /  ",font=fon,command=lambda:press("/"))
btndiv.grid(row=4, column=3, ipadx=38, ipady=13)
btnclear=Button(root, text="clear",font=fon,command=clear)
btnclear.grid(row=5, column=0, columnspan=4, ipadx=237,ipady=13)
```

Output:



5) Program to Create Arcade

```
import arcade
w=600
h=600
arcade.open_window(w,h, "Arcade Example")
arcade.set_background_color(arcade.color.WHITE)
arcade.start_render()

x=300
y=300
radius=200
arcade.draw_circle_filled(x, y, radius, arcade.color.YELLOW)

x=370
y=350
radius=20
arcade.draw_circle_filled(x,y, radius, arcade.color.BLACK)

x = 230
y = 350
radius = 20
arcade.draw_circle_filled(x,y,radius,arcade.color.BLACK)

x=300
y=240
w=120
h=100
start_angle=200
end_angle=350
```

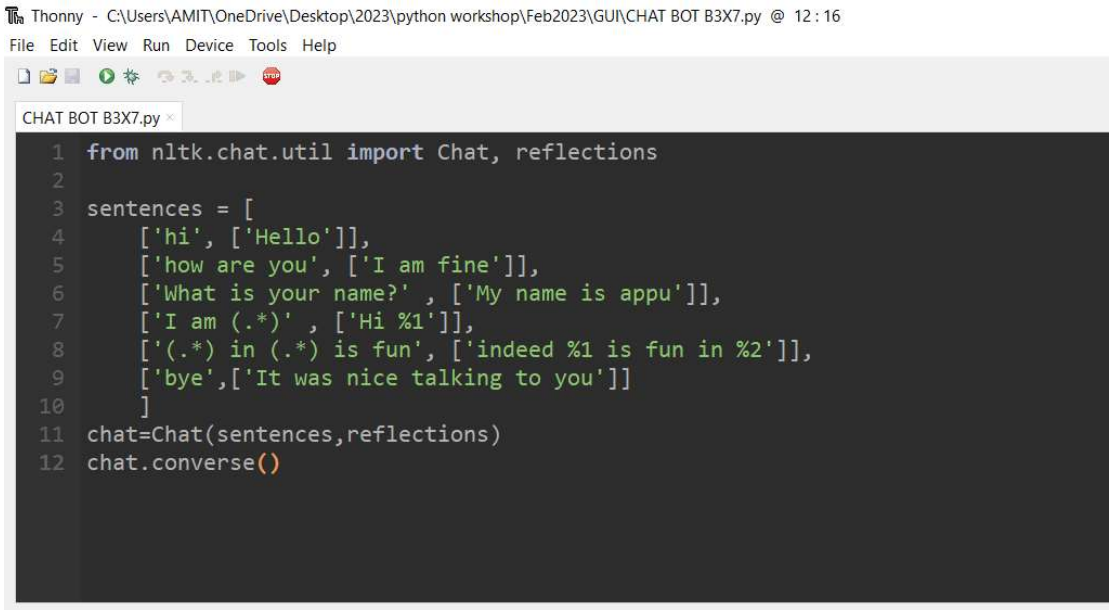
```
arcade.draw_arc_outline(x, y, w, h, arcade.color.BLACK, start_angle, end_angle, 20)

arcade.finish_render()
arcade.run()
```

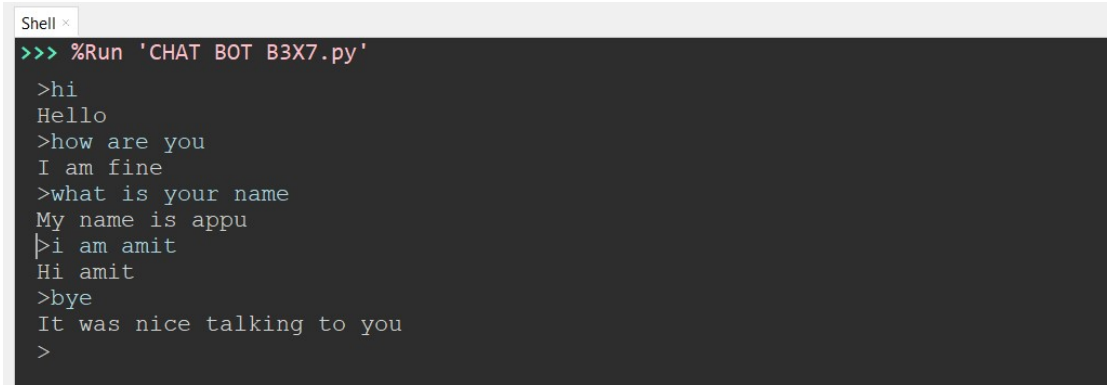
Output:

6) Program to create Chatbot

```
from nltk.chat.util import Chat, reflections
sentences = [
   ['hi', ['Hello']],
   ['how are you', ['I am fine']],
   ['What is your name?' , ['My name is appu']],
   ['I am (.*)' , ['Hi %1']],
   ['(.*) in (.*) is fun', ['indeed %1 is fun in %2']],
   ['bye',['It was nice talking to you']]
   ]
chat=Chat(sentences,reflections)
chat.converse()
```

Output:





7) Program to create Hangman Game

```
import random
from colorama import *
name = input("Enter your name :")
print(name+", Welcome to Hangman Game...")
WORDS=["Siem", "Computers", "Banana", "Orange", "Mathematics",
"Congratulations", "Hangman", "Landlord"]
word=random.choice(WORDS)
attempts = 5
guesses = ""
while attempts>0:
    wrong=0
    for char in word:
        if char.lower() in guesses:
            print(char)
        else:
            print("*")
            wrong+=1
```

```
    if wrong==0:
        print(Fore.GREEN, name,"Congratulations, you win....")
        print(Fore.YELLOW, "The correct word is :", word, Fore.WHITE)
        break

    guess=input("guess a character :")
    guesses +=guess.lower()

    if guess not in word:
        attempts -= 1
        print(Fore.BLUE, "You have ", attempts, "attempts left", Fore.WHITE)

    if attempts==0:
        print(Fore.RED, name, "you loose", Fore.WHITE)
```

Output:

```
>>> %Run hangman1.py
 Enter your name :Amit
 Amit, Welcome to Hangman Game...
 *
 *
 *
 *
 guess a character :s
  You have  4 attempts left
 S
 *
 *
 *
 guess a character :i
 S
 i
 *
 *
 guess a character :e
 S
 i
 e
 *
```

```
guess a character :m
S
i
e
m
 Amit Congratulations, you win....
 The correct word is : Siem
>>>
```

8) Write a program to print leap year and days in a month

```
m = int(input("Enter year: "))
n = int(input("Enter month: "))
leap = 0
if (m%4==0):
   print(m," is a leap year")
   leap = 1
elif (m%400==0):
   print(m," is a leap year")
   leap = 1
else:
   print(m," is not a leap year")

monthno = [1,3,5,7,8,10,12]
if n in monthno:
   days = 31;
else:
   if n==2:
      if leap==1:
         days=29;
      else:
         days=28;
```

```
    else:
        days = 30;
print("Number of days in month: ", days)
```

Output:

```
>>> %Run leapyear.py
 Enter year: 1988
 Enter month: 2
 1988  is a leap year
 Number of days in month:  29
>>>
```

Write the following number pattern programs :

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

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

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```
3

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

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

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```
6

```
11
11 22
11 22 33
11 22 33 44
11 22 33 44 55
```
7

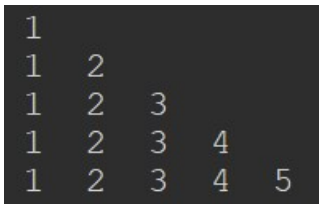
```
11 22 33 44 55
11 22 33 44
11 22 33
11 22
11
```
8

```
        1
      2 1 2
    3 2 1 2 3
  4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```
9

Program 1:

```
for x in range(1,6):
    for y in range(1,x+1):
        print(y, " " , end = "")
    print()
```

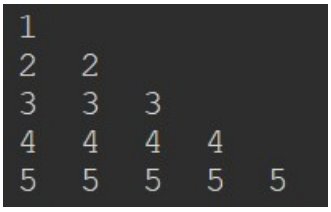
Output:



Program 2:

```
for x in range(1,6):
    for y in range(1,x+1):
        print(y, " " , end = "")
    print()
```

Output:

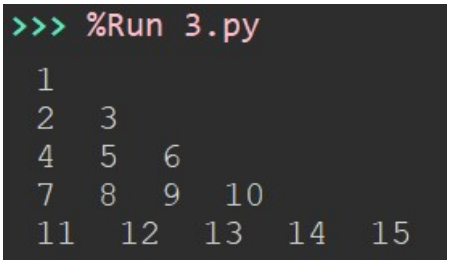


Program 3:

```
for x in range(1,6):
    for y in range(1,x+1):
```

```
        print(y, " " , end = "")
    print()
```

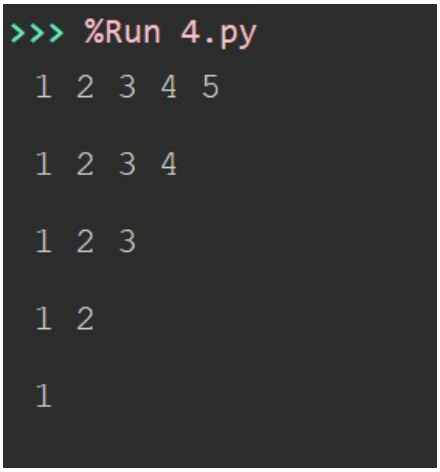
Output:



Program 4:

```
for x in range(6,1,-1):
    for y in range(1,x):
        print(y,"", end="")
    print("\n")
```

Output:



Program 5:

```
for x in range(0,5):
    for y in range(5,x,-1):
        print(y-x," ", end="")
    print()
```

Output:

```
>>> %Run 5.py
 5  4  3  2  1
 4  3  2  1
 3  2  1
 2  1
 1
```

Program 6:

```
for x in range(5,0,-1):
    for y in range(0,x):
        print(x," ", end="")
    print()
```

Output:

```
>>> %Run 6.py
 5  5  5  5  5
 4  4  4  4
 3  3  3
 2  2
 1
```

Program 7:

```
for x in range(1,6):
    for y in range(0,x):
        print(11*x,"",end="")
    print("\n")
```

Output:

```
>>> %Run 7.py
 11

 22 22

 33 33 33

 44 44 44 44

 55 55 55 55 55
```

Program 8:

```
for x in range(6,1,-1):
    for y in range(1,x):
        print(11*y,"",end="")
    print("\n")
```

Output:

```
>>> %Run 8.py
 11 22 33 44 55

 11 22 33 44

 11 22 33

 11 22

 11
```

Program 9:

```
for x in range(1,6):
    for k in range(x,5):
        print(end="\t")
    for y in range(x,0,-1):
        print(y, end="\t")
    for y in range(2, x+1):
        print(y, end="\t")
    print("\n")
```

Output:

```
>>> %Run 9.py
                        1
                    2   1   2
                3   2   1   2   3
            4   3   2   1   2   3   4
        5   4   3   2   1   2   3   4   5

>>>
```

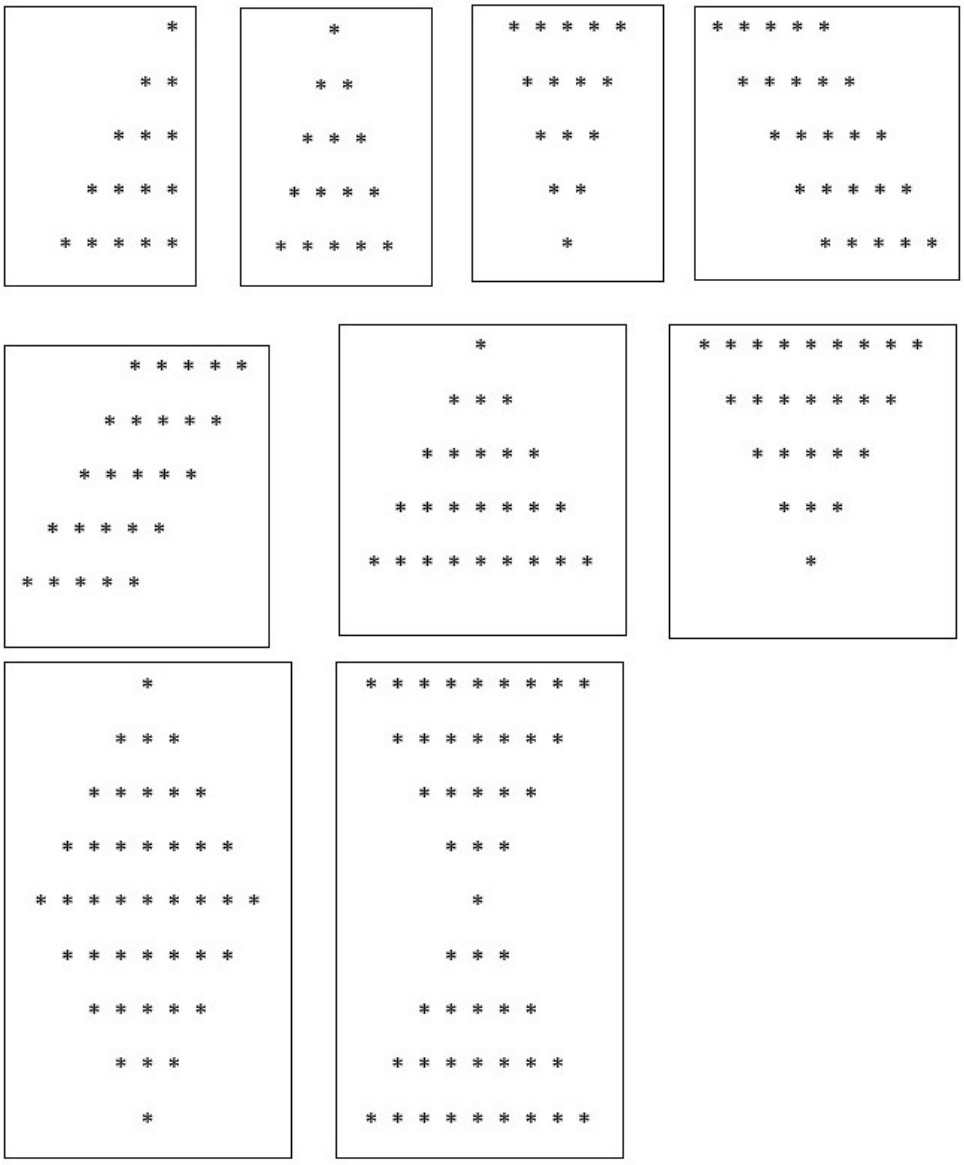
Write the following pattern programs :

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

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

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

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



1.

```
for x in range(1,6):
    for y in range(1,6):
        print("*"," ", end="")
```

```
    print("\n")
```

Output:

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

2. Do it yourself

3.

```
for x in range(6,1,-1):
    for y in range(1,x):
        print("*"," ", end="")
    print("\n")
```

Output:

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

4.

```
for x in range(1,6):
    for y in range(0,x):
```

```
    print("\t", end="")
  for z in range(6,x,-1):
    print("*\t", end="")
  print("\n")
```

Output:

```
        *       *       *       *       *

                *       *       *       *

                        *       *       *

                                *       *

                                        *
```

5.

```
for x in range(1,6):
    for y in range(6,x,-1):
        print("\t",end="")
    for z in range(0,x):
        print("*\t", end="")
    print("\n")
```

Output:

```
                                *

                        *       *

                *       *       *

        *       *       *       *
```

```
*       *       *       *       *
```

6.

```
for x in range(1,6):
    for k in range(5,1,-1):
        print(" ",end="")
    for y in range(0,x):
        print("*",end="")
    print("\n\n")
```

Output:

```
*

*     *

*     *       *

*     *       *       *

*     *       *       *       *
```

7. Do it yourself

8.

```
for x in range(1,6):
    for y in range(1,x):
        print(" ", end="")
    for k in range(1,6):
        print("*", end="")
    print("\n\n")
```

Output:

```
>>> %Run 8.py
 *****

  *****

   *****

    *****

     *****
```

9.

```
for x in range(1,6):
    for y in range(5,x,-1):
        print(" ", end="")
    for k in range(1,6):
        print("*", end="")
    print("\n\n")
```

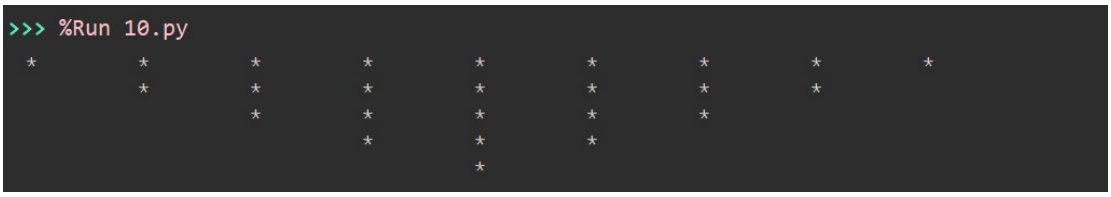
Output:

```
>>> %Run 9.py
     *****

    *****

   *****

  *****

 *****
```

10.

```
for i in range(5,0,-1):
    for k in range(5,i,-1):
        print("\t",end="")
    for j in range(0,2*i-1):
        print("*\t", end = "")
    print("\r\r")
```

Output:



11. Do it yourself

12.

```
num =9
for i in range(1,num+1):
  i = i - (num//2 +1)
  if i<0:
    i = -i
  print("\t" * i + "*\t" * (num - i*2) + "\t"*i)
```

Output:

```
>>> %Run 12.py
                                *
                        *       *       *
                *       *       *       *       *
        *       *       *       *       *       *       *
*       *       *       *       *       *       *       *       *
        *       *       *       *       *       *       *
                *       *       *       *       *
                        *       *       *
                                *
```

13.

```
for x in range(1,5):
    for k in range(1,x):
        print("\t", end="")
    for y in range(x*2-1, 10):
        print("*\t", end="")
    print("\n")
for x in range(1,6):
    for k in range(x,5):
        print("\t",  end="")
```

Output:

```
>>> %Run 13.py
*       *       *       *       *       *       *       *       *

        *       *       *       *       *       *       *

                *       *       *       *       *

                        *       *       *

                                *

                        *       *       *

                *       *       *       *       *

        *       *       *       *       *       *       *

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

Other Programs

1. Write a program in python to print the factorial of a number.

Program:

```
x = int(input("Enter the number of which factorial has to find out: "))
fact = 1
for i in range(1, x+1):
    if(i!=x):
        print(i, "X ", end="")
    else:
        print(i, "=", end="")
    fact = fact*i
print(fact)
```

Output:

```
>>> %Run 1.py
 Enter the number of which factorial has to find out: 5
 1 X 2 X 3 X 4 X 5 =120
```

2. Write a program in python to print series:

1 / 1 ! +2 / 2 ! +3 / 3 ! ----

Program:

```
x = int(input("Enter a number:"))
sum = 0
```

```
for i in range(1, x+1):
    fact =1
    for j in range (1, i+1):
        fact = fact*j
    sum = sum + i/ fact
    if i!=x:
        print(i, "/", i, "!", "+", end="")
    else:
```

Output:

```
>>> %Run 2.py
 Enter a number:3
 1 / 1 ! +2 / 2 ! +3 / 3 ! =2.5
>>> %Run 2.py
 Enter a number:6
 1 / 1 ! +2 / 2 ! +3 / 3 ! +4 / 4 ! +5 / 5 ! +6 / 6 ! =2.7166666666666663
```

3. Write a program to print following series :

1...11...111...1111...11111...

Program:

```
x = int(input("Enter a number: "))
sum=0
for i in range(0,x+1):
    sum=sum+(10**i)
    print(sum,end="")
    print("...",end="")
```

Output:

```
>>> %Run 3.py
 Enter a number: 4
 1...11...111...1111...11111...
```

4. Write a program to print following series:

1+1 ^ 2 / 2 +1 ^ 3 / 3+ ----

```
x = int(input("Enter a number: "))
a = int(input("Enter a: "))
z = x+1
k=0
sum=0
print("1+", end="")
for i in range(2,z):
    k=0
    for j in range(i, i+1):
        print(a, "^",i,"/",i,"",end="")
        k=(a**j)/i
    sum=sum+k
    if(i<z-1):
        print("+", end="")
print("=",sum+1)
```

Output:

```
>>> %Run 4.py
 Enter a number: 3
 Enter a: 1
 1+1 ^ 2 / 2 +1 ^ 3 / 3 = 1.8333333333333333
```

Editors Details

ISBN: 978-81-964018-7-0

Prof Amit Kumar Mishra pursued his Bachelor of Engineering from Shantilal Shah Engineering College (Government College) Bhavnagar Gujarat in Electronics and Communication branch & did his M.E in Communication Engineering from MIT Aurangabad Maharashtra. He is pursuing PhD from department of Science and Technology of Savitribai Phule Pune University Pune in Electronics & Telecommunication domain. Presently he is working as Assistant Professor in E & TC Department of Sandip Institute of Engineering and Management Nashik Maharashtra India. He has total 14 years of teaching experience. He received grant of Rupees One Lakh from BCUD Savtribai Phule Pune University Pune. He has total 20 publications in various International & National Conferences and Journals. He has membership of different professional bodies like ISTE, IEI, INAAR, I2OR. Also, Prof Amit Mishra has received three Awards (one International and two National), he has published four books and two chapters. One of his chapter named "Health Detection System for COVID-19 Patients Using IoT" (Book Title - Medical Imaging and Health Informatics) is published in Scopus. Prof Amit Mishra got felicitated by Hon'ble Education minister of Maharastra Mr Chandrakant Patil during an event "Engineering Talent Search 2022" for being Jury member of the event.

Dr. Dipak Pandurang Patil, currently working as a Professor and Principal at Sandip Institute of Engineering and Management Nashik Maharashtra India. In addition, he is Dean of International Affairs. He has more than 20 years of experience in teaching. He has completed PhD in Electronics and Telecommunication Engineering from Sant Gadge Baba Amravati University in Electronics & Telecommunication Engineering. He has more than 30 publications to his credit in reputed International Journals and Conferences along with five book chapters and three Indian patent on his research. He has membership of different National & International professional bodies like IEEE, ISTE, EAI and IAENG. He has delivered expert talks and sessions in various Institutes and conferences, and served as session chair and designated reviewer in reputed international conferences. In addition to this, he has also received awards at state and national level. He has participated in ERASMUS+ mobility project and received prestigious YOUTHPASS of European Commission.

Dr. Tushar H. Jaware holds a bachelor's degree in electronics and telecommunication engineering from North Maharashtra University, Jalgaon. He further pursued a master's degree in digital electronics and obtained a Ph.D. in medical image processing from Sant Gadge Baba Amravati University, Amravati. Currently serving as the Dean of Research and Development at the R. C. Patel Institute of Technology in Shirpur, Maharashtra, India, Dr. Jaware possesses over 18 years of invaluable teaching experience. He is widely recognized as a Ph.D. Supervisor in electronics engineering at North Maharashtra University, Jalgaon, and Dr. Babasaheb Ambedkar Technological University, Lonere. Furthermore, he has contributed as a Member of the Board of Studies in electronics and telecommunication engineering at North Maharashtra University, Jalgaon. Demonstrating his innovative prowess, Dr. Jaware holds three international and national patents, along with six copyrighted works. His research findings have been published in 62 esteemed research papers featured in renowned international journals and conferences. His expertise in the field has garnered invitations as a Plenary Speaker to numerous prestigious events. Dr. Jaware has been bestowed with several accolades, including the Loksatta Tarun Tejankit Award in 2019 and the GIS Young Innovator and Researchers Award (Central India) in 2018, presented by JSR Laboratory, Pune, in collaboration with the Asian Society for Scientific Research. He also received the esteemed 'Bright Researcher Award' from the International Institute of Organized Research in 2017. Additionally, Dr. Jaware has been honored with 12 awards recognizing his outstanding research and academic contributions by various societies. Notably, he has secured research grants from AICTE under the SPICES scheme and under the Unnat Bharat Abhiyaan initiative.

Xoffencer International Publication
838- Laxmi Colony. Dabra,
Gwalior, Madhya Pradesh, 475110
www.xoffencerpublication.in

MRP: ₹ 450/-
ISBN
9789394707955