Department of AIT-Computer Science & Engineering

SUBJECT: Programming in Python Lab

(22CSH-287)

B.E. 4th Semester

(Branch: CSE-AIML)

Prepared By: Siddharth Kumar (E12853)

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

LAB MANUAL

Chandigarh University

Gharuan, Mohali

Table of Contents

Vision of Chandigarh University

To be globally recognized as a Centre of Excellence for Research, Innovation, Entrepreneurship and disseminating knowledge by providing inspirational learning to produce professional leaders for serving the society.

Mission of Chandigarh University

Providing world class infrastructure, renowned academicians and ideal environment for Research, Innovation, Consultancy and Entrepreneurship relevant to the society.

Offering programs & courses in consonance with National policies for nation building and meeting global challenges.

Designing Curriculum to match international standards, needs of Industry, civil society and for inculcation of traits of Creative Thinking and Critical Analysis as well as Human and Ethical values.

Ensuring students delight by meeting their aspirations through blended learning, corporate mentoring, professional grooming, flexible curriculum and healthy atmosphere based on co-curricular and extra-curricular activities.

Creating a scientific, transparent and objective examination/evaluation system to ensure an ideal certification.

Establishing strategic relationships with leading National and International corporates and universities for academic as well as research collaborations.

Contributing for creation of healthy, vibrant and sustainable society by involving in Institutional Social Responsibility (ISR) activities like rural development, welfare of senior citizens, women empowerment, community service, health and hygiene awareness and environmental protection.

Vision of the Department

To be recognized as a centre of excellence for Computer Science & Engineering education and research, through effective teaching practices, hands-on training on cutting edge computing technologies and excellence in innovation, for creating globally aware competent professionals with strong work ethics whom would be proficient in implementing modern technology solutions and shall have entrepreneurial zeal to solve problems of organizations and society at large.

Mission of the Department

M1: To provide relevant, rigorous and contemporary curriculum and aligned assessment system to ensure effective learning outcomes for engineering technologies.

M2: To provide platform for industry engagement aimed at providing hands-on training on advanced technological and business skills to our students.

M3: To provide opportunities for collaborative, interdisciplinary and cutting-edge research aimed at developing solutions to real life problems.

M4: To imbibe quest for innovation, continuous learning and zeal to pursue excellence through hard work and problem-solving approach.

M5: To foster skills of leadership, management, communication, team spirit and strong professional ethics in all academic and societal endeavours of our students.

Program Education Objectives

PE01: To be able to explore areas of research, technology application & innovation and make a positive impact in different types of institutional settings such as corporate entities, government bodies, NGOs, inter-government organizations, & start-ups.

PEO2: To be able to design, and implement technology and computing solutions to the organizational problems, effectively deploy knowledge of engineering principles, demonstrate critical thinking skills & make the intellectual connections between quantitative and qualitative tools, theories and context to solve the organizational problems.

PEO3: To be able to work with, lead & engage big and small teams comprising diverse people in terms of gender, nationality, region, language, culture & beliefs. To understand stated and unstated differences of views, beliefs & customs in diverse & inter disciplinary team settings **PEO4**: To be able to continuously learn and update one's knowledge, engage in lifelong learning habits and acquire latest knowledge to perform in current work settings.

PEO5: To continuously strive for justice, ethics, equality, honesty, and integrity both in personal and professional pursuits. Able to understand and conduct in a way that is responsible and respectful.

Program Specific Outcomes (PSOs)

PSO1: The graduate student shall be able to analyse and apply data mining techniques and tools to extract detailed information from big datasets.

PSO2: The graduate student shall be able to acquire efficiency in using the big data tools and technologies to handle, analyse, design, develop and test data.

PSO3: The graduate student shall be able to demonstrate big data analysis skills for effective interpretation and decision making to solve real world problems.

Program Outcomes (POs)

PO1: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2: Identify, formulate, review research literature and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal, and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

PO5: Create, select, and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context to technological change.

Course Description

Course Introduction: Welcome to "Programming in Python"! This course is designed to be your gateway into the dynamic and versatile world of Python programming. Whether you're a beginner taking your first steps into the realm of coding or an experienced developer looking to expand your skill set, this course has something for everyone. Python is not just a programming language; it's a powerful tool that has become integral to diverse fields such as web development, data science, artificial intelligence, and more. Known for its readability, simplicity, and vast community support, Python is the perfect language to kickstart your coding journey.

What You'll Learn: Throughout this course, you'll embark on a hands-on journey that covers the fundamentals of Python programming. From understanding basic syntax to creating web applications and delving into data manipulation, each module is crafted to build a strong foundation and practical skills that you can apply in real-world scenarios.

Course Highlights: Problem-Solving Adventures: Tackle real-world problems, sharpen your analytical skills, and learn the art of crafting elegant solutions using Python.

Data Magic: Dive into the realm of data manipulation and analysis, uncovering the power of Python libraries that make handling data a breeze.

Web Wonders: Explore the world of web development with Python. Create dynamic websites, understand the magic behind server-side scripting, and master the art of crafting APIs.

Collaboration Chronicles: Learn the essentials of collaborative coding using Git. Work seamlessly in a team, understand version control, and contribute to shared code repositories.

Prerequisites: No prior programming experience is required, and whether you're a student, professional, or an enthusiast eager to learn, this course is designed to accommodate all levels of learners. All that's needed is your curiosity and a passion for diving into the exciting realm of Python programming.

Let's Get Started: Get ready for an engaging and rewarding experience as we embark on this Pythonic journey together. Whether you dream of building the next big app, analysing vast

datasets, or simply love the art of coding, "Programming in Python" will equip you with the skills and confidence to turn those dreams into reality. Let the coding adventure begin!

Course Objectives

The course aims to:

1. To be able to understand programming skills in Python.
2. To be able to comprehend various operators of Python Programming Language.
3. To demonstrate about Python data structures like Lists, Tuples, Sets and Dictionaries
4. To understand about Functions, Modules and Regular Expressions in Python Programming.
5. To develop the ability to write applications in Python.

Course Outcomes

CO1: Identify and interpret the basics syntax of Python Programming Language and be fluent in the use of Python.

CO2: Express proficiency in the handling of conditional statements, loops, strings and functions.

CO3: Understanding the data structures like Lists, Tuples, Sets and Dictionaries in programming paradigms of Python Programming Language.

CO4: Ability to create practical and contemporary applications using Functions, Abstract Data Types and Modules in Python.

CO5: Implementation of Python based applications using file input and output operations in Python Programming.

SN		Course Name: Programming in Python	L	T	P	S	C	CH	Course Type
1	22CSH-287	Course Coordinator: Siddharth Kumar	2	0	4	0	4	6	Core
PREREQUISITE		Nil							
CO-REQUISITE		Nil							
ANTI-REQUISITE		Nil							

CO-PO & CO-PSO Articulation Matrix

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	1	2	3	2	2	-	-	1	2	1	1	3	3	1
CO2	3	1	3	1	2	2	-	-	1	2	1	1	3	3	1
CO3	3	2	2	3	3	2	-	-	1	1	1	1	3	3	1
CO4	2	2	3	3	3	2	-	-	1	2	1	1	2	2	1
CO5	2	2	2	3	2	2	-	-	1	2	1	2	2	2	1

Syllabus

Theory

Unit-1		Contact Hours: 12 hours
Introduction	The Programming Cycle for Python, Elements of Python, Type Conversion.	
Basics	Expressions, Assignment Statement, Arithmetic Operators, Operator Precedence, Boolean Expression.	
Conditional Statements	Conditional Statement in Python (if-else statement, its working and execution), Nested-if Statement and Else if Statement in Python, Expression Evaluation & Float Representation.	
Loops	Purpose and working of Loops, While Loop including its Working, For Loop, Nested Loops, Break and Continue.	
Unit-2		**Contact Hours: 12 Hours**
Function	Parts of a Function, Execution of a Function, Keyword and Default Arguments, Scope Rules.	
Strings	Length of the string and perform concatenation and Repeat operations, Indexing and Slicing of Strings.	
Python Data Structure	Tuples, Unpacking Sequences, Lists, Mutable Sequences, List Comprehension, Sets, Dictionaries.	
Higher Order Functions	Treat functions as first-class Objects, Lambda Expressions.	
Unit-3		**Contact Hours: 12 Hours**
Abstract Data Types	Abstract data types and ADT interface in Python Programming.	
Classes	Class definition and other operations in the classes, Special Methods (such as _init_, _str_, comparison methods and Arithmetic methods etc.), Class Example, Inheritance, Inheritance and OOP.	
Iterators & Recursion	Recursive Fibonacci, Tower of Hanoi.	
Search	Simple Search and Estimating Search Time, Binary Search and Estimating Binary Search Time.	
Sorting & Merging	Selection Sort, Merge List, Merge Sort, Higher Order Sort.	
File I/O	File input and output operations in Python Programming Exceptions and Assertions.	
Modules	Introduction and Importing Modules	

Practical

Lab Number	Unit Number	Topic/Sub-Topic
1	1	Demonstrate about Basics of Python Programming.
2	1	Demonstration of various operators used in Python with suitable example programs. i) Arithmetic Operators, ii) Relational Operators, iii) Assignment Operator, iv) Logical Operators, and v) Bit wise Operators
3	1	Write a program for the following Conditional Statements in Python with suitable examples. i) if statement ii) if else statement iii) if – elif – else statement
4	1	Write a program for the following Iterative Statements in Python with suitable examples. i) while loop ii) for loop
5	2	Write a program for the following functions/methods which operates on lists in Python with suitable examples: i) list( ) ii) len( ) iii) count( ) iv) index ( ) v) append( ) vi) insert( ) vii) extend() viii) remove( ) ix) pop( ) x) reverse( ) xi) sort( ) xii) copy( )
6	2	Write a Python program to demonstrate various ways of accessing the string. i) By using Indexing (Both Positive and Negative) ii) By using Slice Operator
7	2	Write a Python program to demonstrate the different ways of creating tuple objects with suitable example.
8	3	Write a Python program to return multiple values at a time using a return statement.
9	3	Implement the following Searching and Sorting techniques in Python by using functions. i) Linear Search ii) Binary Search iii) Selection Sort iv) Merge Sort v) Quick Sort
10	3	Python program to perform read and write operations on a file.

Lab Requirements

Programming in Python generally has modest hardware requirements, making it accessible for a wide range of computers. Here are the recommended computer specifications for programming in Python:

Processor (CPU): A multi-core processor, such as an Intel Core i5 or AMD Ryzen 5, is recommended. Python is not particularly CPU-intensive for basic programming tasks, so a mid-range processor is usually sufficient.

Memory (RAM): At least 8 GB of RAM is recommended for smooth Python programming, especially when working with data-intensive tasks or larger projects. Having more RAM can be beneficial for handling larger datasets and complex applications.

Storage (Hard Drive or SSD): A solid-state drive (SSD) is preferable for faster read and write speeds, providing a smoother overall experience. However, a traditional hard drive (HDD) with sufficient capacity (at least 256 GB) can also work well.

Operating System: Python is cross-platform and supports Windows, macOS, and Linux. Choose an operating system based on your preference and requirements. Many developers prefer using Linux or macOS for Python development, but Windows is also widely used.

Development Environment: Choose a code editor or integrated development environment (IDE) for Python programming. Popular choices include Visual Studio Code, PyCharm, Jupyter Notebooks, and Atom.

Graphics Processing Unit (GPU): For general Python programming, a dedicated GPU is not a strict requirement. However, if you plan to work on machine learning or data science projects that involve GPU acceleration, having a compatible GPU (NVIDIA CUDA-enabled, for example) can be beneficial.

Internet Connection: A stable internet connection is recommended for downloading packages, libraries, and documentation. Additionally, online resources and collaboration platforms are essential for Python developers.

Version Control System: Install a version control system like Git on your machine. This is crucial for tracking changes in your code, collaborating with others, and managing different versions of your projects.

Text Editors and IDEs: Set up your preferred text editor or integrated development environment (IDE) for coding. Ensure it supports Python syntax highlighting and provides features for efficient coding and debugging.

Python Interpreter: Install the latest version of Python from the official website (https://www.python.org/). Many developers use virtual environments to manage dependencies and isolate project-specific packages.

These specifications are general guidelines, and the actual requirements may vary based on the complexity of your projects. Python's flexibility allows it to run on a wide range of systems, making it accessible for beginners and professionals alike.

Assessment Pattern

For the **Theory Courses**, the performance of students is evaluated as follows:

Components	Continuous Internal Assessment (CAE)	Semester End Examination (SEE)
Marks	40	60
Total Marks	100	

Frequency for assessment tools for theory classes

Sr. No.	Type of Assessment Task	Weightage of Actual conduct	Frequency of Task	Final Weightage of internal assessment (Prorated Marks)
1	Assignment*	10 Marks for each assignment	One per unit	10 Marks
2	Time bound Surprise Test	12 Marks for each test	One per unit	4 Marks
3	Quiz	4 marks for each quiz	Two per unit	4 Marks
4	Mid-Semester Test**	20 Marks for one MST	Two per semester	20 Marks
5	Presentation***	NA	As Applicable	Non-Graded Engagement Task
6	Homework	NA	One per lecture topic (of 2 questions)	Non-Graded Engagement Task
7	Discussion Forum	NA	One per chapter	Non-Graded Engagement Task
8	Attendance and Engagement Score	NA	NA	2 Marks

*Every teacher should include one innovation based (Video/Simulation/LTI Based) assignment for the

students other than only essay type questions.

**Mid-Semester Test to be conducted physical in examination halls. But in case the COVID scenario extends, then it has to be conducted in Online Model via proctored examination software.

**This category may be graded in case of Seminar/Project type courses.

For the **Practical Courses**, the performance of students is evaluated as follows:

Components	Continuous Internal Assessment (CAE)	Semester End Examination (SEE)
Marks	60	40
Total Marks	100	

Frequency for assessment tools for practical classes (since 2020)

Sr. No.	Type of Assessment Task	Weightage of Actual conduct	Frequency of Task	Final Weightage of internal assessment (Prorated Marks
1	Practical Project/ Workshop/ Assignment /Classroom Learning	20 marks for each Project / Assignment / Experiment	8-10 Practical Project / Assignments/ Experiments	40 marks
2	Mid-Term Test	20 marks	1 per Semester	12 Marks
3	Portfolio/ Discussion Forum/Short Digital Assignment	4 marks for each task	1 per semester	4 marks
4	Presentation	--	As Applicable	Non-Graded: Engagement Task
5	Attendance and BB Engagement Score	--	NA	4 marks

Announcements

Dear Students,

Get ready to embark on a coding adventure like never before! We're thrilled to announce a brand-new course, "Python Programming for Students," designed to ignite your passion for coding and unleash your creative potential.

Why Python? Python is not just a programming language; it's a superpower that opens doors to endless possibilities. Whether you're a beginner or an experienced coder, Python's simplicity and versatility make it the perfect language to learn. From creating web applications to diving into data science, Python is your ticket to the world of technology.

What to Expect:

- 🚀 **Hands-on Coding:** Dive right into practical coding exercises and real-world projects that bring concepts to life.
- 🌐 **Web Development:** Create your first dynamic website and understand the magic behind server-side scripting.
- 📊 **Data Magic:** Uncover the power of Python in data manipulation, analysis, and visualization.
- 🤝 **Collaborative Coding:** Learn to work seamlessly in a team using Git, an essential skill for modern developers.

Who Can Join: Whether you're a computer science enthusiast, a curious learner, or someone eager to explore the world of programming, this course is tailored for you. No prior coding experience required – just bring your enthusiasm and curiosity!

When Does It Start: Mark your calendars! The course kicks off in upcoming semester. Stay tuned for the detailed schedule and exciting session lineup.

Get Ready to Code! We can't wait to see you in the virtual classroom, where Python magic happens. Whether you dream of building the next big app, analysing vast datasets, or simply love the art of coding, this course is your gateway to a world of endless possibilities.

UNIT 1

Experiment 1.1

Mapped Course Outcomes - CO1

CO1: Identify and interpret the basics syntax of Python Programming Language and be fluent in the use of Python.

Aim: Demonstrate about Basics of Python Programming.

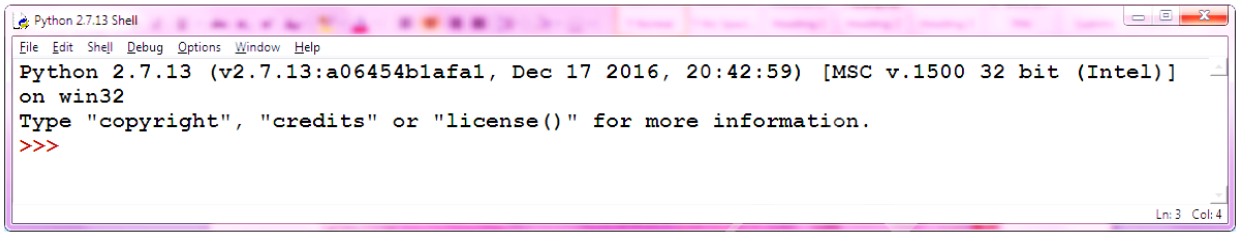
Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

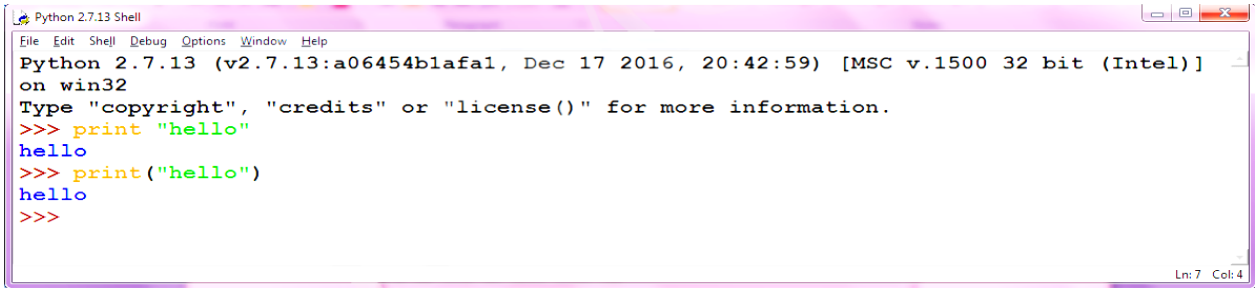
1 Installation of Python Tool
2 Install the required python libraries
3 Writing code as per the aim
4 Simulating the code
5 Checking for errors
6 Analyzing the results

Code:

Running Python Interpreter: Python comes with an interactive interpreter. When you type python in your shell or command prompt, the python interpreter becomes active with a >>> (REPL) and waits for your commands.



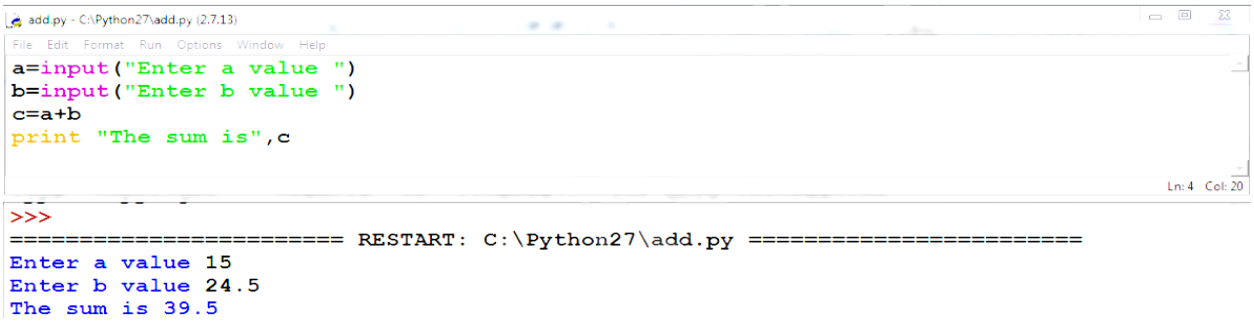
Now you can type any valid python expression at the prompt. Python reads the typed expression, evaluates it and prints the result



Running Python Scripts in IDLE (Integrated Development and Learning Environment): IDLE is the standard Python development environment. Its name is an acronym of "Integrated Development Environment". It works well on both Unix and Windows platforms. It has a Python shell window, which gives you access to the Python interactive mode. It also has a file editor that lets you create and edit existing Python source files. Goto File menu click on New File (CTRL+N) and write the code and save add.py

```
a=input ("Enter a value ")
b=input ("Enter b value ")
c=a+b
print "The sum is", c
```

Then run the program by pressing F5 or Run ==> Run Module.



Running Python scripts in Command Prompt: Before going to run python27 folder in the command prompt. Open your text editor, type the following text and save it as hello.py.

```
print "hello"
```

In the command prompt, and run this program by calling python hello.py. Make sure you change to the directory where you saved the file before doing it.

Outcome of the Experiment:

• Learn to installation of python and relate tools with libraries and about their Interpreter.

• Learn to running Python scripts in Command Prompt as well as in Python Scripts in IDLE (Integrated Development and Learning Environment).

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=pfPCV7DXc5w&pp=ygUycHl0aG9uIHNoZWxsIGluc3RhbGxhdGlvbiBpbiB3aW5kb3dzIDExIGluIGVuZ2xpc2g%3D

Other resource materials or Web Links:

https://www.python.org/downloads/windows/

Questions to be asked for Viva Voce:

Q1. Why is Python often referred to as an interpreted language?
Q2. What is the purpose of the Python shell, and how does it differ from a script?
Q3. Can you explain the significance of the PATH environment variable during Python installation on Windows 11?
Q4. What steps are involved in installing Python on Windows 11 using the official Python installer?

Q5. How does the installation of Python on Windows 11 differ from other operating systems like Linux or macOS?

Q6. What are the advantages of using a virtual environment in Python, and how can you create one on Windows 11?

Q7. Explain the role of the Python launcher in managing multiple Python installations on a Windows system.

Q8. What is the purpose of the "Add Python to PATH" checkbox during the Python installation process?

Q9. Describe the steps to check if Python is installed correctly on a Windows 11 system.

Q10. How do you upgrade an existing Python installation to a newer version on Windows 11?

Q11. Can you install multiple versions of Python on the same Windows 11 machine? If so, how would you manage them?

Q12. Explain the concept of a Python virtual environment and why it is recommended for Python development.

Q13. What is the significance of the Python executable (python.exe) and how is it utilized in Windows 11?

Q14. How would you troubleshoot common issues encountered during the Python installation process on Windows 11?

Q15. Can you briefly discuss alternative methods for installing Python on Windows 11, such as using package managers like Anaconda or Chocolatey?

Experiment 1.2

Mapped Course Outcomes - CO1

CO1: Identify and interpret the basics syntax of Python Programming Language and be fluent in the use of Python.

Aim: Demonstration of various operators used in Python with suitable example programs. i) Arithmetic Operators, ii) Relational Operators, iii) Assignment Operator, iv) Logical Operators, and v) Bit wise Operators.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Create script file to write code
3 Install the required python libraries
4 Writing code as per the aim
5 Simulating the code one by one
6 Checking for errors
7 Analyzing the results

Code:

Operator is a symbol which can perform specified operation on operands.

1. Arithmetic Operators: The arithmetic operations are the basic mathematical operators which are used in our daily life. Mainly it consists of seven operators.

i. Addition operator --> '+'

ii. Subtraction operator --> '-'

iii. Multiplication operator --> '*'

iv. Normal Division operator --> '/'

v. Modulo Division operator --> '%'

vi. Floor Division operator --> '//'

vii. Exponential operator (or) power operator --> '**'

i. Addition Operator: Generally, addition operator is used to perform the addition operation on two operands. But in python we can use addition operator to perform the concatenation of strings, lists and so on, but operands must of same datatype.

```
In [1]:

x = 2
y = 3
print("ADDITION RESULT : ", x + y)

ADDITION RESULT :  5

In [2]:

x = 2
y = 3.3
print("ADDITION RESULT : ", x + y) # both float and int type are accept

ADDITION RESULT :  5.3

In [3]:

x = 2.7
y = 3.3
print("ADDITION RESULT : ", x + y) # both float type are accept

ADDITION RESULT :  6.0

In [4]:

x = "2"
y = 3
print("ADDITION RESULT : ", x + y) #str type and int can't be added.

---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-4-d873d6fd7998> in <module>
      1 x = "2"
      2 y = 3
----> 3 print("ADDITION RESULT : ", x + y) #str type and int can't be adde
d.

TypeError: can only concatenate str (not "int") to str

In [5]:

x = "2"
y = "3"
print("ADDITION RESULT : ", x + y) # concatenation will take place

ADDITION RESULT :  23
```

In [6]:

```
x = "2"
y = 4.8
print("ADDITION RESULT : ", x + y) # float type and str typr can't be added.
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-6-32bf23d43c09> in <module>
      1 x = "2"
      2 y = 4.8
----> 3 print("ADDITION RESULT : ", x + y) # float type and str typr can't
be added.

TypeError: can only concatenate str (not "float") to str
```

In [7]:

```
x = 2
y = bool(4.8)
print("ADDITION RESULT : ", x + y) #here bool(4.8) returns True i.e, 1
```

```
ADDITION RESULT :  3
```

In [8]:

```
x = "2"
y = bool(4.8)
print("ADDITION RESULT : ", x + y) #bool type cant be concatenated.
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-8-aa2b47f2b5f5> in <module>
      1 x = "2"
      2 y = bool(4.8)
----> 3 print("ADDITION RESULT : ", x + y) #bool type cant be concatenate
d.

TypeError: can only concatenate str (not "bool") to str
```

In [9]:

```
x = "2"
y = str(bool(4.8))
print("ADDITION RESULT : ", x + y)
#bool returns 1 generally but we converted into str then it gives True
```

```
ADDITION RESULT :  2True
```

In [9]:

```
x = "2"
y = str(bool(4.8))
print("ADDITION RESULT : ", x + y)
#bool returns 1 generally but we converted into str then it gives True
```

```
ADDITION RESULT :  2True
```

In [10]:

```
x = "2"
y = str(complex(4.8))        #Here both strings so concatenation will take place
print("ADDITION RESULT : ", x + y)
```

```
ADDITION RESULT :  2(4.8+0j)
```

In [11]:

```
x = 2
y = complex(4.8)
print("ADDITION RESULT : ", x + y)
# here both are int type so addtion will take place
```

```
ADDITION RESULT :  (6.8+0j)
```

ii. Subtraction Operator: Generally, subtraction operator is used to perform the subtraction operation on two operands.

In [12]:

```
a = 30
b = 10
print("Subtraction result : ",a-b)
```

```
Subtraction result :  20
```

In [13]:

```
a = 30
b = "10"
print("Subtraction result : ",a-b)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-13-f0fd62944ccb> in <module>
      1 a = 30
      2 b = "10"
----> 3 print("Subtraction result : ",a-b)

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

In [14]:

```
a = "30"
b = "10"
print("Subtraction result : ",a-b)
# can not perform subtraction on str type operands.
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-14-0bebbed27be9> in <module>
      1 a = "30"
      2 b = "10"
----> 3 print("Subtraction result : ",a-b)
      4 # can not perform subtraction on str type operands.

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

In [15]:

```
a = 20
b = 10.00
print("Subtraction result : ",a-b)
```

```
Subtraction result :  10.0
```

In [16]:

```
a = 20
b = bool(10)
print("Subtraction result : ",a-b)
```

```
Subtraction result :  19
```

iii. Multiplication operator: Generally, multiplication operator is used to perform the multiplication operation on two operands. But in python we can use multiplication operator to perform the repetition of strings, lists and so on, but operands must belong to same datatype.

```
In [17]:

num1 = 23
num2 = 35
print("MULTIPLICATION RESULT : ",num1 * num2)

MULTIPLICATION RESULT :  805

In [18]:

num1 = 23
num2 = 35.0
print("MULTIPLICATION RESULT : ",num1 * num2)

MULTIPLICATION RESULT :  805.0

In [19]:

num1 = "23"
num2 = 5
print("MULTIPLICATION RESULT : ",num1 * num2) # 23 string will prints 5 times

MULTIPLICATION RESULT :  2323232323

In [20]:

num1 = "23"
num2 = "5"
print("MULTIPLICATION RESULT : ",num1 * num2)

---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-20-e4135d9e3a29> in <module>
      1 num1 = "23"
      2 num2 = "5"
----> 3 print("MULTIPLICATION RESULT : ",num1 * num2)

TypeError: can't multiply sequence by non-int of type 'str'

In [21]:

l = "(1,2,3,4)"
print(float(l * 5))

---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-21-18109e54b2f8> in <module>
      1 l = "(1,2,3,4)"
----> 2 print(float(l * 5))

ValueError: could not convert string to float: '(1,2,3,4)(1,2,3,4)(1,2,3,
4)(1,2,3,4)(1,2,3,4)'

In [22]:

l = "123"
print(float(l * 4))
#initially it will prints string 5 times and converts it into float

123123123123.0
```

In [23]:

```
l = "123"
b = 2.3
print("MULTIPLICATION RESULT : ", l * b)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-23-e235870edcad> in <module>
      1 l = "123"
      2 b = 2.3
----> 3 print("MULTIPLICATION RESULT : ", l * b)

TypeError: can't multiply sequence by non-int of type 'float'
```

In [24]:

```
l = "123"
b = bool(2.3)
print("MULTIPLICATION RESULT : ", l * b)
```

```
MULTIPLICATION RESULT :  123
```

In [25]:

```
l =[1,2,3]
m = [2,4,5]
print(l * m) # multiplication of two list data types is not possible
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-25-309b92e03dcb> in <module>
      1 l =[1,2,3]
      2 m = [2,4,5]
----> 3 print(l * m) # multiplication of two list data types is not possib
le

TypeError: can't multiply sequence by non-int of type 'list'
```

In [26]:

```
l = (5,6,7)
m = (1,2,3)
print(l* m) # multiplication of two tuple data types is not possible
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-26-91a31577591d> in <module>
      1 l = (5,6,7)
      2 m = (1,2,3)
----> 3 print(l* m) # multiplication of two tuple data types is not possib
le

TypeError: can't multiply sequence by non-int of type 'tuple'
```

In [27]:

```
l = bool(1)
m = bool(4657)
print(l * m) # as bool returns 1 it prints only one time
```

```
1
```

In [28]:

```
l = bool()
m = bool(123456789)
print(l*m) # As bool doesn't contain any value it consider as zero.
```

```
0
```

In [29]:

```
l = str(bool([1,2,3]))
m = 99
print(l*m)
```

```
TrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTr
ueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrue
TrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTr
ueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrue
TrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTrueTr
ueTrueTrueTrueTrueTrue
```

In [30]:

```
l = bool([1,2,3])
m = 99
print(l*m)
```

```
99
```

iv. Division Operator: Generally, division operator is used to perform the division operation on two operands. It returns the result in float type.

In [31]:

```
a = 3
b = 45
print("Division result : ", a/ b) # returns float value
```

```
Division result :  0.06666666666666667
```

In [32]:

```
a = 3
b = "45"
print("Division result : ", b / a)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-32-1b2bbedeebd4> in <module>
      1 a = 3
      2 b = "45"
----> 3 print("Division result : ", b / a)

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

In [33]:

```
a = 3
b = 45.0000
print("Division result : ", b / a)
```

```
Division result :  15.0
```

In [34]:

```
a = 3
b = bool(0.0000)
print("Division result : ", a / b)
```

```
---------------------------------------------------------------------------
-
ZeroDivisionError                         Traceback (most recent call las
t)
<ipython-input-34-854e10cbf4f9> in <module>
      1 a = 3
      2 b = bool(0.0000)
----> 3 print("Division result : ", a / b)

ZeroDivisionError: division by zero
```

In [35]:

```
a = 3
b = complex((90))
print("Division result : ", a / b)
```

```
Division result :  (0.03333333333333333+0j)
```

In [36]:

```
a = [1,2,3]
b = [7,8,9]
print("Division result : ", a / b)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-36-8289b4627a90> in <module>
      1 a = [1,2,3]
      2 b = [7,8,9]
----> 3 print("Division result : ", a / b)

TypeError: unsupported operand type(s) for /: 'list' and 'list'
```

In [37]:

```
a = (1,2,3)
b = (1,2,3)
print("Division result : ", a / b)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-37-f02db8ba9671> in <module>
      1 a = (1,2,3)
      2 b = (1,2,3)
----> 3 print("Division result : ", a / b)

TypeError: unsupported operand type(s) for /: 'tuple' and 'tuple'
```

In [38]:

```
a = {1,2,3}
b = {1,2,3}
print("Division result : ", a / b)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-38-cd4ea53f676a> in <module>
      1 a = {1,2,3}
      2 b = {1,2,3}
----> 3 print("Division result : ", a / b)

TypeError: unsupported operand type(s) for /: 'set' and 'set'
```

In [39]:

```
l = bool()
m = bool(9)
print(l / m)
```

```
0.0
```

v. Modulo Division Operator: It returns reminder.

In [40]:

```
a = 3
b = 4
print(a%b)
print(b%a)
```

```
3
1
```

vi. Floor Division Operator: Suppose 10.3 is there, what is the floor value of 10.3?

Answer is 10

What is the ceil value of 10.3?

Answer is 11.

In [41]:

```
print(10/2)
```

```
5.0
```

In [42]:

```
print(10/3)
```

```
3.3333333333333335
```

→ If you want to get integer value as result of division operation, you need to make use of floor division(//)

→ operator.

→ floor division(//) operator meant for integral arithmetic operations as well as floating point arithmetic

→ operations.

→ The result of floor division(//) operator can be always floor value of either integer value or float value

→ based on your arguments.

→ If both arguments are 'int' type, then the result is 'int' type.

→ If at least one of the argument is float type, then the result is also float type.

```
In [43]:
print(10//2)
5

In [44]:
print(10/3)
3.3333333333333335

In [45]:
print(10.0/3)
3.3333333333333335

In [46]:
print(10.0//3)
3.0

In [47]:
print(10//3)
3

In [48]:
print(10.0//3.0)
3.0

In [49]:
print(20/2)
print(20.5/2)
print(20//2)
print(20.5//2)
print(30//2)
print(30.0//2)
10.0
10.25
10
10.0
15
15.0
```

vii. Power Operator or Exponential Operator: It is used for calculation of power of any number.

In [50]:

```
print(10**2) # meaning of this is 10 to the power 2
print(3**3)
```

```
100
27
```

2. Relational Operators (or) Comparison Operators: Following are the relational operators used in Python:

1. Less than (<)

2. Greater than (>)

3. Less than or Equal to (<=)

4. Greater than or Equal to (>=)

In [51]:

```
a = 10
b = 20
print('a < b is', a<b)
print('a <= b is', a<=b)
print('a > b is', a>b)
print('a >= b is', a>=b)
```

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

How to know the Unicode or ASCII value of any character?

- By using **ord()** function, we can get the ASCII value of any character.

In [52]:

```
print(ord('a'))
print(ord('A'))
```

```
97
65
```

If you know the ASCII value and to find the corresponding character, you need to use the chr() function.

In [53]:

```
print(chr(97))
print(chr(65))
```

```
a
A
```

In [54]:

```
s1 = 'karthi' # ASCII value of 'a' is 97
s2 = 'sahasra' # ASCII value of 'b' is 98
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
True
True
False
False
```

In [55]:

```
s1 = 'karthi'
s2 = 'karthi'
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
False
True
False
True
```

In [56]:

```
s1 = 'karthi'
s2 = 'Karthi'
print(s1<s2)
print(s1<=s2)
print(s1>s2)
print(s1>=s2)
```

```
False
False
True
True
```

iii) We can apply relational operators even for boolean types also.

In [57]:

```
print(True > False)
print(True >= False) # True ==> 1
print(True < False) # False ==> 0
print(True <= False)
```

```
True
True
False
False
```

```
In [58]:

print(10 > 'karthi')

---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-58-e2ae37134b58> in <module>
----> 1 print(10 > 'karthi')

TypeError: '>' not supported between instances of 'int' and 'str'


In [60]:

a = 10
b = 20
if a>b:
    print('a is greater than b')
else:
    print('a is not greater than b')

a is not greater than b
```

Equality Operators: Equality operators are used to check whether the given two values are equal or not. The following are the

equality operators used in Python.

1. Equal to (==)
2. Not Equal to (!=)

```
print(10==20)
print(10!=20)

False
True
```

3. Assignment Operators: We can use assignment operator to assign value to the variable.

x = 2 # We can combine assignment operator with some other operator to form compound assignment operator.

```
a,b=23,43 # a =23 b = 43
c = 50 if a>b else 100
print(c)

100
```

4. Logical operators: Following are the various logical operators used in Python.

1. and
2. or
3. not

You can apply these operators for Boolean types and non-Boolean types, but the behaviour is different.

For Boolean types:

and ==> If both arguments are True then only result is True

or ==> If at least one argument is True then result is True

not ==> complement

i) 'and' Operator for Boolean type: If both arguments are True then only result is True

```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

```
True
False
False
False
```

ii) 'or' Operator for Boolean type: If both arguments are True then only result is True.

```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

```
True
True
True
False
```

iii) 'not' Operator for Boolean type: Complement (or) Reverse

```
In [69]:

print(not True)
print(not False)
```

```
False
True
```

Now we will try to develop a small authentication application with this knowledge.

→ we will read user name and password from the keyboard.

→ if the user name is karthi and password is sahasra, then that user is valid user otherwise invalid user.

```
userName = input('Enter User Name : ')
password = input('Enter Password : ')
if userName == 'karthi' and password == 'sahasra':
    print('valid User')
else:
    print('invalid user')
```

```
Enter User Name : karthi
Enter Password : sahasra
valid User
```

```
userName = input('Enter User Name : ')
password = input('Enter Password : ')
if userName == 'karthi' and password == 'sahasra':
    print('valid User')
else:
    print('invalid user')
```

```
Enter User Name : ECE
Enter Password : CSE
invalid user
```

5. Bitwise Operators: We can apply these operators bit by bit. These operators are applicable only for int and Boolean types. By mistake if we are trying to apply for any other type then we will get Error. Following are the various bitwise operators used in Python:

1. Bitwise and (&)
2. Bitwise or (|)
3. Bitwise ex-or (^)
4. Bitwise complement (~)
5. Bitwise left shift Operator (<>)

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=WS4zSIXJqmw&pp=ygUjb3BlcmF0b3JzIHVzZWQgaW4gUHl0aG9uIGluIGVuZ2xpc2g%3D

Other resource materials or Web Links:

https://www.w3schools.com/python/python_operators.asp

Questions to be asked for Viva Voce:

Q1. What are the arithmetic operators in Python?

Q2. How does the exponentiation operator (**) work in Python?

Q3. Explain the purpose of the += and *= assignment operators.

Q4. What is the difference between = and == in Python?

Q5. List the comparison operators in Python.

Q6. How does the "not equal to" operator (!=) differ from the "equal to" operator (==)?

Q7. Explain the use of the logical AND, OR, and NOT operators.

Q8. How does short-circuiting work with logical operators in Python?

Q9. What are bitwise operators, and how are they used in Python?

Q10. Explain the XOR (^) operator in bitwise operations.

Q11. What is the purpose of the in and not in operators in Python?

Q12. What is the purpose of the floor division operator in Python?

Q13. How does it differ from regular division (/)?

Experiment 1.3

Mapped Course Outcomes – CO2

CO2: Express proficiency in the handling of conditional statements, loops, strings and functions.

Aim: Write a program for the following Conditional Statements in Python with suitable examples. i) if statement ii) if else statement iii) if – elif – else statement".

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create a script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

i) if statement:

Syntax:

```
if condition:
    statement 1
    statement 2
    statement 3
```

```
if 10<20:
    print('10 is less than 20')
print('End of Program')
```

```
10 is less than 20
End of Program
```

ii) if else statement:

Syntax:

```
if condition:
        Action 1
else:
        Action 2
```

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

```
name = input('Enter Name : ')
if name == 'Karthi':
    print('Hello Karthi! Good Morning')
else:
    print('Hello Guest! Good Morning')
print('How are you?')
```

```
Enter Name : Karthi
Hello Karthi! Good Morning
How are you?
```

iii) if – elif – else statement:

Syntax:

```
if condition 1:
        Action 1
elif condition 2:
        Action 2
else:
        Action 3 (Default Action)
```

if condition 1 is true then Action-1 will be executed, if condition 2 is true then Action-2 will be executed otherwise Action-3 will be executed.

```
brand=input("Enter Your Favourite Brand:")
if brand=="RC":
    print("It is childrens brand")
elif brand=="KF":
    print("It is not that much kick")
elif brand=="FO":
    print("Buy one get Free One")
else :
    print("Other Brands are not recommended")
```

```
Enter Your Favourite Brand:RC
It is childrens brand
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=wIXfXYf17ok&pp=ygUrQ29uZGl0aW9uYWwgU3RhdGVtZW50cyBpbiBQeXRob24gaW4gZW5nbGlzaA%3D%3D

Other resource materials or Web Links:

https://www.w3schools.com/python/python_conditions.asp

Questions to be asked for Viva Voce:

Q1. What is a conditional statement in Python, and why is it essential in programming?

Q2. Explain the syntax of the if statement in Python. How is it structured?

Q3. How does Python handle indentation in conditional statements? Why is it crucial for the code's functionality?

Q4. What is the purpose of the else statement in Python's conditional constructs? Provide an example.

Q5. Describe the role of the elif statement in Python. When would you use it, and how does it differ from else?

Q6. Can you have multiple elif statements following an if statement? If so, what purpose do they serve?

Q7. What is the significance of the colon (:) in Python conditional statements?

Q8. Explain the concept of a nested if statement. Provide an example to illustrate its usage.

Q9. How does short-circuit evaluation apply to Python's conditional statements? Provide an example.

Q10. Discuss the importance of logical operators (and, or, not) in constructing complex conditional expressions.

Q11. What happens if the condition in an if statement is not met, and there is no else block?

Q12. How can you use conditional statements to check if a variable is within a specific range? Provide an example.

Q13. Explain the ternary conditional expression in Python. How is it different from an if statement?

Q14. Discuss the scenario in which you might use a switch-case statement in other programming languages. Does Python support a switch-case statement?

Experiment 1.4

Mapped Course Outcomes – CO2

CO2: Express proficiency in the handling of conditional statements, loops, strings and functions.

Aim: "Write a program for the following Iterative Statements in Python with suitable examples. i) while loop ii) for loop".

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create a script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

i) while loop:

Syntax:

```
while condition:
        body
```

Code to print numbers from 1 to 10 by using while loop.

```
x=1
while x <=10:
    print(x,end=' ')
    x=x+1
```

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

Code to display the sum of first 'n' numbers.

```
n=int(input("Enter number:"))
sum=0
i=1
while i<=n:
    sum=sum+i
    i=i+1
print("The sum of first",n,"numbers is :",sum)
```

ii) for loop:

Syntax:

```
for x in sequence:
        body
```

- Where, 'sequence' can be string or any collection.
- Body will be executed for every element present in the sequence.

Code to print characters present in the given string.

```
s="Sahasra"
for x in s :
    print(x)
```

```
S
a
h
a
s
r
a
```

```
s="Sahasra"
for x in s :
    print(x,end='\t')
```

```
S       a       h       a       s       r       a
```

Code to print characters present in string index wise.

```
s=input("Enter some String: ")
i=0
for x in s :
    print("The character present at ",i,"index is :",x)
    i=i+1
```

```
Enter some String: Karthikeya
The character present at  0 index is : K
The character present at  1 index is : a
The character present at  2 index is : r
The character present at  3 index is : t
The character present at  4 index is : h
The character present at  5 index is : i
The character present at  6 index is : k
The character present at  7 index is : e
The character present at  8 index is : y
The character present at  9 index is : a
```

Code to print Hello 10 times.

```
s = 'Hello'
for i in range(1,11):
    print(s)
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

```
s = 'Hello'
for i in range(10):
    print(s)
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Code to display numbers from 0 to 10.

```
for i in range(0,11):
     print(i,end=' ')
```

```
0 1 2 3 4 5 6 7 8 9 10
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=njXjk-t863E&pp=ygUpaXRlcmF0aXZlIHN0YXRlbWVudHMgaW4gcHl0aG9uIGluIGVuZ2xpc2g%3D

Other resource materials or Web Links:

https://www.geeksforgeeks.org/loops-and-control-statements-continue-break-and-pass-in-python/

Questions to be asked for Viva Voce:

Q1. What is the purpose of using iterative statements in programming, and why are they essential in various algorithms?

Q2. Explain the syntax of the while loop in Python. How is it structured, and what role does the condition play?

Q3. Describe the difference between the while and for loops in Python. When would you choose one over the other?

Q4. How does Python handle indentation in loops, and why is it a crucial aspect of the language's syntax?

Q5. Discuss the concept of an infinite loop. How can you avoid creating one unintentionally?

Q6. What is the purpose of the break statement in a loop? Provide an example where it might be used.

Q7. Explain the use of the continue statement in a loop. How does it affect the flow of the loop execution?

Q8. Discuss the importance of the range () function in the context of for loops. How does it work, and how can it be utilized?

Q9. How do you iterate over elements in a list using a for loop in Python? Provide an example.

Q10. Explain the concept of nested loops. Can you provide an example of when and why nested loops might be used?

Q11. How does the else block in a loop function in Python? When is it executed?

Q12. Discuss the term "iteration" in the context of loops. How many iterations will a loop perform if the condition is true?

Q13. Explain the role of the pass statement in a loop. When might it be useful, and how does it impact the loop's behavior?

Q14. How can you use the enumerate() function in a for loop to access both the index and value of elements in an iterable?

UNIT 2

Experiment 2.1

Mapped Course Outcomes – CO3

CO3: Understanding the data structures like Lists, Tuples, Sets and Dictionaries in programming paradigms of Python Programming Language.

Aim: Write a program for the following functions/methods which operates on lists in Python with suitable examples: i) list( ) ii) len( ) iii) count( ) iv) index ( ) v) append( ) vi) insert( ) vii) extend() viii) remove( ) ix) pop( ) x) reverse( ) xi) sort( ) xii) copy( ).

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

i) list( ): Used to create list (like array) in Python

```
l=list(range(0,10,2))
print(l)
# Not working in jupyter notebook but works in any standard editor
```

ii) len( ): It returns the number of elements present in the list.

```
n=[10,20,30,40]
print(len(n))
```

```
4
```

iii) count( ): It returns the number of occurrences of specified item in the list.

```
n=[1,2,2,2,2,3,3]
print(n.count(1))
print(n.count(2))
print(n.count(3))
print(n.count(4))
```

```
1
4
2
0
```

iv) index ( ): It returns the index of first occurrence of the specified item.

```
n=[1,2,2,2,2,3,3]
print(n.index(1)) # 0
print(n.index(2)) # 1
print(n.index(3)) # 5
print(n.index(4))
```

```
0
1
5
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-4-b7a2785b40f1> in <module>
      3 print(n.index(2)) # 1
      4 print(n.index(3)) # 5
----> 5 print(n.index(4))

ValueError: 4 is not in list
```

v) append( ): We can use append() function to add item at the end of the list. By using this append function, we always add an element at last position.

```
list=[]
list.append("A")
list.append("B")
list.append("C")
print(list)
```

```
['A', 'B', 'C']
```

```
list=[]
for i in range(101):
    if i%10==0:
        list.append(i)
print(list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Other way

```
list= []
for i in range(0,101,10):
    list.append(i)
print(list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

vi) insert( ): It is used to insert item at specified index position.

```
n=[1,2,3,4,5]
n.insert(1,888)
print(n)
```

```
[1, 888, 2, 3, 4, 5]
```

Differences between append() and insert()

append()	insert()
In List when we add any element it will come in last i.e. it will be last element.	In List we can insert any element in particular index number

vii) extend(): If we want to add all items of one list to another list,we use extend() method.

L1.extend(L2) #All items present in L2 will be added to L1

```
order1=["Chicken","Mutton","Fish"]
order2=["RC","KF","FO"]
order1.extend(order2)
print(order1)
print(order2)
```

```
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']
['RC', 'KF', 'FO']
```

viii) remove( ): We can use this function to remove specified item from the list. If the item present multiple times, then only first occurrence will be removed.

```
n=[10,20,10,30]
n.remove(10)
print(n)
```

```
[20, 10, 30]
```

ix) pop( ): It removes and returns the last element of the list. This is only function which manipulates list and returns some element.

```
n=[10,20,30,40]
print(n.pop())
print(n.pop())
print(n)
```

```
40
30
[10, 20]
```

Differences between remove() and pop()

remove()	pop()
1) We can use to remove special element from the List.	1) We can use to remove last element from the List.
2) It can't return any value.	2) It returned removed element.
3) If special element not available then we get VALUE ERROR.	3) If List is empty then we get Index Error.

x) reverse( ): It is used to reverse the order of elements in the list.

```
n=[10,20,30,40]
n.reverse()
print(n)
```

```
[40, 30, 20, 10]
```

xi) sort( ): In list by default insertion order is preserved. If you want to sort the elements of list according to default natural sorting order then we should go for sort() method.

For numbers ==> default natural sorting order is Ascending Order
For Strings ==> default natural sorting order is Alphabetical Order

```
n=[20,5,15,10,0]
n.sort()
print(n)
```

```
[0, 5, 10, 15, 20]
```

```
s=["Dog","Banana","Cat","Apple"]
s.sort()
print(s)
```

```
['Apple', 'Banana', 'Cat', 'Dog']
```

Note: To use sort() function, compulsory list should contain only homogeneous elements, otherwise we will get TypeError.

xii) copy( ): To copy the data

```
x=[10,20,30,40]
y=x.copy()
y[1]=777
print(x) # [10,20,30,40]
print(y) # [10,777,30,40]
```

```
[10, 20, 30, 40]
[10, 777, 30, 40]
```

Relevant demonstrative Video Link:

https://youtu.be/o-tKvGHjrvo

Other resource materials or Web Links:

https://www.geeksforgeeks.org/list-methods-in-python-set-2-del-remove-sort-insert-pop-extend/

Questions to be asked for Viva Voce:

Q1.What is the purpose of the list() function in Python?

Q2.Can you provide an example of using list() to convert a tuple to a list?

Q3. How does the len() function work with lists in Python?

Q4. What does len() return for an empty list?

Q5. Explain the use of the count() method in Python lists.

Q6. Provide an example where count() is used to find the occurrences of a specific element.

Q7. Describe the purpose of the index() method in Python lists.

Q8. What happens if the specified element is not present in the list when using index()?

Q9. How does the append() method modify a list in Python?

Q10. Provide an example demonstrating the use of append() to add elements to a list.

Q11. Explain the functionality of the insert() method in Python lists.

Q12. Can you provide an example of using insert() to add an element at a specific index?

Q13. What is the purpose of the extend() method in Python lists?

Q14. How does extend() differ from append()?

Q15. Describe how the remove() method is used to eliminate elements from a list.

Q16. What happens if the specified element is not present in the list when using remove()?

Q17. Explain the role of the pop() method in Python lists.

Q18. Can you provide an example of using pop() to remove an element at a specific index?

Q19. How does the reverse() method alter the order of elements in a list?

Q20. Can you reverse a list without using the reverse() method?

Q21. Discuss the purpose of the sort() method in Python lists.

Q22. Provide an example of using sort() to arrange elements in ascending order.

Q23. What is the function of the copy() method for Python lists?

Q24. How does copy() differ from simple assignment (=) when creating a new list?

Experiment 2.2

Mapped Course Outcomes – CO3

CO3: Understanding the data structures like Lists, Tuples, Sets and Dictionaries in programming paradigms of Python Programming Language.

Aim: Write a Python program to demonstrate various ways of accessing the string. i) By using Indexing (Both Positive and Negative) ii) By using Slice Operator.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

String: Any sequence of characters within either single quotes or double quotes is considered as a String.

Syntax:

```
s='karthi'

s="karthi"
```

Note: In most of other languages like C, C++, Java, a single character with in single quotes is treated as char data type value. But in Python we are not having char data type. Hence it is treated as String only.

```
ch = 'a'
print(type(ch))
```

```
<class 'str'>
```

How to access characters of a String?

We can access characters of a string by using the following ways.

1. By using index
2. By using slice operator

1. By using index:

Python supports both +ve and -ve index.

+ve index means left to right(Forward direction).

-ve index means right to left(Backward direction).

```
s = 'Karthi'
print(s[0])
print(s[5])
print(s[-1])
print(s[19])
```

```
K
i
i
```

2. Accessing characters by using slice operator:

String slice means a part of the string (i.e, Sub string).

Syntax:

string_Name [beginindex:endindex:step]

Here, i. beginindex: From where we have to consider slice(substring)

ii. endindex: We have to terminate the slice(substring) at endindex-1

iii. step: incremented / decremented value

Note: Slicing operator returns the sub string form beginindex to endindex - 1

If we are not specifying begin index then it will consider from beginning of the string.

If we are not specifying end index then it will consider up to end of the string.

The default value for step is 1

```
s = 'abcdefghijk'
print(s[2:7])
```

```
cdefg
```

```
s = 'abcdefghijk'
print(s[:7])
```

```
abcdefg
```

```
s = 'abcdefghijk'
print(s[2:])
```

```
cdefghijk
```

```
s = 'abcdefghijk'
print(s[:])
```

```
abcdefghijk
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=cPypu6RLijA

Other resource materials or Web Links:

https://www.geeksforgeeks.org/how-to-index-and-slice-strings-in-python/

Questions to be asked for Viva Voce:

Q1.Describe the purpose of the index() method in Python lists.

Q2.What happens if the specified element is not present in the list when using index()?

Q3.How does the append() method modify a list in Python?

Q4.Provide an example demonstrating the use of append() to add elements to a list.

Q5.Explain the functionality of the insert() method in Python lists.

Q6.Can you provide an example of using insert() to add an element at a specific index?

Experiment 2.3

Mapped Course Outcomes – CO3

CO3: Understanding the data structures like Lists, Tuples, Sets and Dictionaries in programming paradigms of Python Programming Language.

Aim: Write a Python program to demonstrate the different ways of creating tuple objects with suitable example.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

1. Tuple is exactly same as List except that it is immutable. i.e., once we creates Tuple object, we cannot perform any changes in that object. Hence Tuple is Read Only Version of List.

2. If our data is fixed and never changes then we should go for Tuple.

3. Insertion Order is preserved.

4. Duplicates are allowed.

5. Heterogeneous objects are allowed.

6. We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.

7. Tuple support both +ve and -ve index. +ve index means forward direction (from left to right) and -ve index means backward direction (from right to left).

8. We can represent Tuple elements within Parenthesis and with comma separator.

Note: Parenthesis are optional but recommended to use.

```
t=10,20,30,40
print(t)
print(type(t))
```

```
(10, 20, 30, 40)
<class 'tuple'>
```

```
t=(10,20,30,40)
print(t)
print(type(t))
```

```
(10, 20, 30, 40)
<class 'tuple'>
```

```
t = ()
print(type(t))
```

```
<class 'tuple'>
```

Note: We have to take special care about single valued tuple. Compulsory the value should ends with comma, otherwise it is not treated as tuple.

```
t=(10)
print(t)
print(type(t))
```

```
10
<class 'int'>
```

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

```
(10,)
<class 'tuple'>
```

Creation of Tuple Objects: We can create empty tuple object

```
t=()
print(t)
print(type(t))
```

```
()
<class 'tuple'>
```

Creation of single valued tuple object:

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

```
(10,)
<class 'tuple'>
```

Creation of multi values tuples & parenthesis are optional:

```
t = 10,20,30
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

We can create a tuple object using tuple() function:

```
list=[10,20,30]
t=tuple(list)
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=SeySX9OwQZQ

Other resource materials or Web Links:

https://www.geeksforgeeks.org/tuples-in-python/

Questions to be asked for Viva Voce:

Q1.Explain the syntax for creating a tuple in Python.

Q2. Can you have a tuple with a single element? If so, how is it defined?

Q3. Discuss the immutability of tuples in Python. How does it impact their usage?

Q4. How are elements accessed in a tuple? Explain the concept of indexing.

Q5. What happens if you try to modify an element in a tuple?

Q6. Compare and contrast tuples and lists in terms of mutability and immutability.

Q7. How is tuple packing and unpacking achieved in Python? Provide examples.

Q8. What is the purpose of the count() method in tuples?

Q9. Explain the functionality of the index() method in tuples.

Q10. Discuss the role of tuples in functions, especially in returning multiple values.

Q11. How can you concatenate two tuples in Python?

Q12. What is the significance of parentheses in a tuple, and can you create an empty tuple?

Q13. How can you convert a list to a tuple in Python?

Q14. Explain the differences between a tuple and a set in Python.

Q15. Discuss the use of tuples as keys in dictionaries.

Q16. How do you iterate through elements in a tuple using a loop?

Q17. What is the purpose of the sorted() function with tuples, and how does it work?

Q18. Can you nest tuples within other tuples in Python? If so, provide an example.

Q19. How does tuple assignment work in Python, and in what scenarios might it be useful?

UNIT 3

Experiment 3.1

Mapped Course Outcomes – CO4

CO4: Ability to create practical and contemporary applications using Functions, Abstract Data Types and Modules in Python.

Aim: Write a Python program to return multiple values at a time using a return statement.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1. Open Python
2. Install the required python libraries
3. Create script file
4. Writing code as per the aim
5. Simulating the code
6. Checking for errors
7. Analyzing the results

Code:

In other languages like C, C++ and Java, function can return at most one value. But in Python, a function can return any number of values.

```
def calc(a,b):        # Here, 'a' & 'b' are called positional arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
a,b,c,d = calc(100,50)      # Positional arguments
print(a,b,c,d)
```

```
150 50 5000 2.0
```

Alternate Way:

```
def calc(a,b): # Positional Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(100,50)
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

```
def calc(a,b):           # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(a = 100, b = 50) # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

```
def calc(a,b): # keyword arguments Arguments
    sum = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return sum,sub,mul,div
t = calc(b = 50, a = 100) # keyword arguments Arguments
for x in t:
    print(x)
```

```
150
50
5000
2.0
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=dH_cKQ__3Gw

Other resource materials or Web Links:

https://www.geeksforgeeks.org/g-fact-41-multiple-return-values-in-python/

Questions to be asked for Viva Voce:

Q1. What is the significance of returning multiple values from a function in Python?

Q2. Explain the syntax for returning multiple values in a Python function.

Q3. How can you assign the returned values from a function to variables?

Q4. Discuss the use of tuples for returning multiple values. Why are tuples commonly employed for this purpose?

Q5. Can you return multiple values without using a tuple? If so, provide an example.

Q6. Explain the concept of tuple unpacking and how it relates to returning multiple values.

Q7. How does the number of variables on the left side of the assignment relate to the number of values returned by a function?

Q8. Discuss the scenario in which you might use a list to return multiple values from a function.

Q9. What happens if the number of variables on the left side of the assignment does not match the number of values returned by a function?

Q10. Compare and contrast the use of tuples and lists for returning multiple values.

Q11. How do you ensure that the order of returned values is maintained when assigning them to variables?

Q12. Discuss the role of dictionaries in returning multiple values from a function.

Experiment 3.2

Mapped Course Outcomes – CO4

CO4: Ability to create practical and contemporary applications using Functions, Abstract Data Types and Modules in Python.

Aim: Implement the following Searching and Sorting techniques in Python by using functions. i) Linear Search ii) Binary Search iii) Selection Sort iv) Merge Sort v) Quick Sort.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

Searching Techniques: Searching is the process of finding the location of a target element among a list of elements. Here, we are going to discuss about the following searching techniques:

i) Linear Search

This is the simplest of all searching techniques. In this method the list is need not be in sorted order. The key element which is to be searched is compared with each element of the list one by one. If match exists, element is found. Otherwise end of list is reached, means search fails, element not found in the list.

Procedure:

1. Read the search element from the user.

2. Compare the search element with the first element in the list.

3. If both are matched, then display "Given element is found!!!" and terminate the function.

4. If both are not matched, then compare search element with the next element in the list.

5. Repeat steps 3 and 4 until search element is compared with last element in the list.

6. If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

```
n=int(input("Enter the number of elements in the array :"))
i=0
flag = 0
a=[i for i in range(n)]
for i in range(0,n):
    a[i]=int(input("Enter the {} element of the array :".format(i+1)))
for i in range(0,n):
    print (a[i])
key=int(input("Enter the Key element to be searched"))
for i in range(0,n):
    if a[i]==key:
        print ("Key element found at",i+1,'Position')
        flag = 1
        break
if flag == 0:
    print("Key element not found !!!")
```

```
Enter the number of elements in the array :3
Enter the 1 element of the array :33
Enter the 2 element of the array :33
Enter the 3 element of the array :33
33
33
33
Enter the Key element to be searched33
Key element found at 1 Position
```

ii) Binary Search

Binary search is another simple searching method. To implement binary search method, the elements of the list must be in sorted order. So, you can apply any one of the sorting techniques

before using the binary search (for example bubble sort). Binary Search method make use of divide and conquer strategy.

Procedure:

1. Binary search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.
2. Binary search looks for a particular item by comparing the middle most item of the collection.
3. If a match occurs, then the index of item is returned.
4. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item.
5. This process continues on the sub-array as well until the size of the sub-array reduces to zero.

```
def binary_search(a,n,key):
    low=0
    high=n-1
    while(low<=high):
        mid=int((low+high)/2)
        if(key==a[mid]):
            return mid
        elif(key<a[mid]):
            high=mid-1
        else:
            low=mid+1
    return -1

n=int(input("Enter the number of elements : "))
a=[i for i in range(n)]
for i in range(0,n):
    a[i]=int(input("Enter the {} element of the array :".format(i+1)))
k=int(input("Enter the key element to be searched :"))
position=binary_search(a,n,k)
if(position!=-1):
    print ("Key element found at ",(position))
else:
    print ("Key element not found !!!")
```

```
Enter the number of elements : 5
Enter the 1 element of the array :23
Enter the 2 element of the array :45
Enter the 3 element of the array :67
Enter the 4 element of the array :79
Enter the 5 element of the array :90
Enter the key element to be searched :79
Key element found at  3
```

iii) Selection Sort

Selection sort is also known as push-down sorting. As the name suggests the first element of the list is selected. It is compared with all the elements. If any element is found to be lesser than the selected element, these two are interchanged. This procedure is repeated till all the elements in the list are sorted.

```
def selectionSort(a):
    for i in range(len(a)):
        least=i
        for k in range(i+1,len(a)):
            if a[k]<a[least]:
                least=k
        temp=a[least]
        a[least]=a[i]
        a[i]=temp

a=[50,30,10,20,40,70,60]
print ("Original list",a)
selectionSort(a)                    # Calling to the function
print("Selection Sort:",a)
```

```
Original list [50, 30, 10, 20, 40, 70, 60]
Selection Sort: [10, 20, 30, 40, 50, 60, 70]
```

iv) Merge Sort

Merge sort was invented by John Von Neumann (1903 - 1957). This sorting method uses divide and conquer method. The basic concept of merge sort is to divide the list into two smaller sub-lists of approximately equal size and continue splitting process until each sub list contains only one element. After this, merge the two parts containing one element into one sorted list and continue merging parts until finally there is only one sorted list. Merge sort is one of the most efficient sorting algorithms.

Procedure:

1. Consider the initial list and divide the list into two sub-lists.

2. Again these sub-lists are divided into many numbers of sub-lists until each and every sub-list contains single element.

3. Combine these sub-lists into sorted order.

4. Finally we will get list of elements in sorted order.

```
def mergeSort(a):
    if len(a)>1:
        mid = len(a)//2
        left= a[:mid]
        right=a[mid:]
        mergeSort(left)
        mergeSort(right)
        i = 0
        j = 0
        k = 0
        while i<len(left) and j<len(right):
            if left[i]<right[j]:
                a[k]=left[i]
                i += 1
            else:
                a[k]=right[j]
                j += 1
            k += 1
        while i<len(left):
            a[k] = left[i]
            i += 1
            k += 1

        while j<len(right):
            a[k] = right[j]
            j += 1
            k += 1
a = list(map(int,input("Enter list Elements: ").split()))
print("Before sorting the elements in the list are: ",a)
mergeSort(a)
print("After sorting the elements in the list are: ",a)
```

```
Enter list Elements: 65 76 23 98 22 11 22
Before sorting the elements in the list are:  [65, 76, 23, 98, 22, 11, 22]
After sorting the elements in the list are:  [11, 22, 22, 23, 65, 76, 98]
```

v) Quick Sort.

This sorting is also called as partition exchange sorting. This method is based on divide and conquer technique. The entire list is divided into various partitions and sorting procedure is applied again and again on the partitions.

Procedure:

1. Divide the collection in two (roughly) equal parts by taking a pseudo-random element and using it as a pivot element.

2. Elements smaller than the pivot get moved to the left of the pivot, and elements larger than the pivot to the right of it.

3. This process is repeated for the collection to the left of the pivot, as well as for the array of elements to the right of the pivot until the whole array is sorted.

```
def partition(a,low,high):
    i = ( low-1 )
    pivot = a[high]
    for j in range(low , high):
        if a[j] <= pivot:
            i = i+1
            temp = a[i]
            a[i] = a[j]
            a[j] = temp
    temp = a[i + 1]
    a[i + 1] = a[high]
    a[high] = temp
    return ( i+1 )
def quickSort(a,low,high):
    if low < high:
        pi = partition(a,low,high)
        quickSort(a, low, pi-1)
        quickSort(a, pi+1, high)

a = list(map(int,input("Enter list Elements: ").split()))
print("Before sorting the elements in the list are: ",a)
quickSort(a,0, len(a)-1)
print("After sorting the elements in the list are: ",a)
```

```
Enter list Elements: 2 3 9 1 4 7 6 8
Before sorting the elements in the list are:  [2, 3, 9, 1, 4, 7, 6, 8]
After sorting the elements in the list are:  [1, 2, 3, 4, 6, 7, 8, 9]
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=d7iGniWrRng

Other resource materials or Web Links:

https://www.geeksforgeeks.org/difference-between-searching-and-sorting-algorithms/

Questions to be asked for Viva Voce:

Q1. What is the significance of returning multiple values from a function in Python?
Q2. Explain the concept of linear search in Python.
Q3. What is the time complexity of linear search?
Q4. Describe how binary search works and when it can be applied.
Q5. What is the prerequisite for using binary search on a list?
Q6. Compare and contrast sequential and binary search.
Q7. In what scenarios is binary search more efficient?
Q8. Provide a step-by-step explanation of implementing binary search in Python.
Q9. What are the key requirements for applying binary search?
Q10. Discuss the concept of interpolation search.
Q11. When is interpolation search preferred over binary search?
Q12. Explain the working of the bubble sort algorithm.
Q13. What is the time complexity of bubble sort?
Q14. Describe the selection sort algorithm and its steps.
Q15. What is the main characteristic of selection sort?
Q16. How does insertion sort function, and what is its key principle?
Q17. Discuss the time complexity of insertion sort.
Q18. Explain the divide-and-conquer strategy in merge sort.
Q19. What is the time complexity of merge sort?
Q20. Describe the quick sort algorithm and its pivot selection strategy.
Q21. Discuss the advantages of quick sort over other sorting algorithms.

Experiment 3.3

Mapped Course Outcomes – CO5

CO5: Implementation of Python based applications using file input and output operations in Python Programming.

Aim: Python program to perform read and write operations on a file.

Apparatus Required: Setting up all the requirements such as internet connectivity, system, python programming language, libraries after installing Python, basic data structures in python.

Step by step procedure:

1 Open Python
2 Install the required python libraries
3 Create script file
4 Writing code as per the aim
5 Simulating the code
6 Checking for errors
7 Analyzing the results

Code:

```
str =input("Enter the data into a file : ")
f1=open('abc.txt','w')
f1.write(str)
f1.close()
f2=open('abc.txt','r')
data=f2.read()
print(data)
f2.close()
```

```
Enter the data into a file : Python Programming Lab Manual
Python Programming Lab Manual
```

```
filename =input("Enter the file name : ")
f1=open(filename,'w')
f1.write('Computer science Engineering\n')
f1.write('Elecronics and Communication Engineering\n')
f1.write('Civil Engineering\n')
f1.close()
f2=open(filename,'r')
data=f2.read()
print(data)
f2.close()
```

```
Enter the file name : RGM
Computer science Engineering
Elecronics and Communication Engineering
Civil Engineering
```

```
filename =input("Enter the file name : ")
f1=open(filename,'w')
f1.write('Computer science Engineering\n')
f1.write('Elecronics and Communication Engineering\n')
f1.write('Civil Engineering\n')
f1.close()
f2=open(filename,'r')
data=f2.readline()
print(data)
f2.close()
```

```
Enter the file name : rgm
Computer science Engineering
```

```
filename =input("Enter the file name : ")
f1=open(filename,'w')
f1.write('Computer science Engineering\n')
f1.write('Elecronics and Communication Engineering\n')
f1.write('Civil Engineering\n')
f1.close()
f2=open(filename,'r')
data=f2.readlines()
print(data)
f2.close()
```

```
Enter the file name : Python
['Computer science Engineering\n', 'Elecronics and Communication Engineeri
ng\n', 'Civil Engineering\n']
```

Relevant demonstrative Video Link:

https://www.youtube.com/watch?v=zEb9zn1RvpM

Other resource materials or Web Links:

https://www.javatpoint.com/os-operations-on-the-file

Questions to be asked for Viva Voce:

Q1. What is the default mode when you open a file using the open() function for reading?

Q2. Explain the purpose of the read() method in Python file handling.

Q3. How does read() differ from readline()?

Q4. What is the significance of the file pointer in file reading operations?

Q5. How can you reset the file pointer to the beginning of the file?

Q6. How do you read a specific number of characters from a file using Python?

Q7. Provide an example of reading the first 50 characters from a file.

Q8. Discuss the use of the with statement when reading from a file.

Q9.Why is it recommended to use the with statement for file handling?

Q10. Explain the difference between reading a file in binary mode ('rb') and text mode ('rt').

Q11. In what scenarios might you choose one mode over the other?

Q12. How do you open a file for writing in Python?

Q13. What happens if the specified file does not exist when opened for writing?

Q14. Discuss the purpose of the write() method in Python file handling.

Q15. Can you write both text and numerical data using the write() method?

Q16. Explain the use of the writelines() method in file writing operations.

Q17. How does writelines() differ from write()?

Q18. How can you append content to an existing file in Python?

Q19. Provide an example of appending text to a file.

Q20. Discuss the difference between the modes 'w' and 'a' when opening a file for writing.

Q21. In what scenarios might you choose 'w' over 'a' or vice versa?

Q22. What is the purpose of the flush() method in file writing?

Q23. When might you use flush() in file operations?

Q24. Explain the use of the seek() method in file handling.

Q25. How can you use seek() to move the file pointer to a specific position?

Q26. Discuss the importance of closing a file after reading or writing operations.

Q27. What happens if you don't close a file after writing to it?

Q28. How can you handle exceptions when working with file operations in Python?

Q29. Provide an example of using a try-except block for file handling.

Q30. What is the purpose of the tell() method in file handling?

Q31. How can you use tell() to determine the current position of the file pointer?

LAB MST

Question Paper (Mapped with COs)

Scheme of Evaluation

1. Set of Unique questions need to be prepared and distributed among the students.

2. Students need to take their seats according to their UIDs in Sequential fashion.

3. No student can change their allocated seat during the practical, to ensure this, you need to make all the computers in working mode. Ask the respective Lab Instructors to install the required Operating System/ Application Software in your guidance.

4. Faculty will visit to respective student desk for the evaluating the Lab components, as mentioned in the bullet 5.

5. Lab Evaluation Components are

a) Worksheet - 8 Marks

b) Conduct - 12 Marks

c) Viva - 10 Marks

Total Marks: 30

Best Practices Adopted

1. Introductory first session about the subject followed by Getting Started Quiz.

2. Post Chapter Discussion Forums.

3. Real World to The Work /Instructor Lead Hands on Learning through Lab work.

4. Self-Assessment Questionnaires. (Through Assignment Task)

5. Peer Teaching

6. Promote Student Engagement through discussions and presentations.

7. Case Studies