A Laboratory Manual for

Python Programming (3151108)

B.E. Semester 5 (E.C. Engineering)

Electronics & Communication Engineering Department

L.D. College of Engineering, Ahmedabad

Directorate of Technical Education, Gandhinagar, Gujarat

Enrolment No. __________________ Batch : ________________

Name of the student: ____

L.D. College of Engineering, Ahmedabad

CERTIFICATE

This is to certify that Mr/Miss ____________________________________Enrollment No. _____________ of B.E. (E.C.) SEM-V of this institute (GTU Code: 028) has satisfactorily completed the practical work of the subject **Python Programming** prescribed by Gujarat Technological University during the academic term July-2025 to October 2025

Date: Signature of the faculty

Vision of Commissionerate of Technical Education, Gujarat State

- To provide globally competitive technical education;
- Remove geographical imbalances and inconsistencies;
- Develop student friendly resources with a special focus on girls' education and support to weaker sections;
- Develop programs relevant to industry and create a vibrant pool of technical professionals.

Vision of the Institute

To contribute for sustainable development of nation through achieving excellence in technical education and research while facilitating transformation of students into responsible citizens and competent professionals.

Mission of the Institute

- To impart affordable and quality education in order to meet the needs of industries and achieve excellence in teaching-learning process.
- To create a conducive research ambience that drives innovation and nurtures research-oriented scholars and outstanding professionals.
- To collaborate with other academic & research institutes as well as industries in order to strengthen education and multidisciplinary research.
- To promote equitable and harmonious growth of students, academicians, staff, society and industries, thereby becoming a center of excellence in technical education.
- To practise and encourage high standards of professional ethics, transparency and accountability.

Vision of the department

To bring out technically competent, socially responsible and ethically sound electronics and communication engineers for the nation.

Mission of the department

- Upgrade learning resources and pedagogical skills for effective teaching-learning process
- Foster research culture and strengthen networking with institutions, industries, research organization and alumni
- Encourage students for professional ethics and social responsibility
- Facilitating students to work on innovative projects related to electronics and communication engineering.

Programme Outcomes (POs)

PO 1	**Engineering knowledge:** Apply the knowledge of mathematics, science, mechanical engineering fundamentals, and specialization to the solution of complex engineering problems.
PO 2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex mechanical engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and mechanical engineering.
PO 3	**Design/development of solutions:** Design solutions for complex mechanical 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.
PO 4	**Conduct investigations of complex problems:** 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.
PO 5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex mechanical engineering activities with an understanding of the limitations.
PO 6	**The engineer and society:** 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.
PO 7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication:** 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.
PO 11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Objectives (PSO)

PSO	PSO statement
PSO 1	Apply fundamental concepts of electronics and communication engineering to identify and solve real life problems.
PSO 2	Utilize analytical and design skills to inculcate habit of working individually and in a group through experiments and project work.
PSO 3	Ability to use modern tools to produce environment friendly ideas and products

Programme Educational Objectives (PEO)

Sr. No.	PEO	PEO statement
1	PEO 1	To make students capable for the post-graduate courses, research, successful career in industries or entrepreneurship in the field of electronics and communication engineering.
2	PEO 2	To design, draft, erect and maintain electronics production.
3	PEO 3	To acquire communication skills to effectively communicate problems and solutions of electronics and communication engineering individually as well as in a team.
4	PEO 4	To be engaged in lifelong learning and adapt changing technological and social needs.
5	PEO 5	To be ethically and professionally sensitive to the consequence of work for promising career.

INDEX

	NodeMCU and send temperature and light data on cloud (Thingspeak, Firebase, Ubidot or any other cloud service)	
14	Student Micro Project	

Course: 31511078 Python Programming

Course outcomes:

CO-1	To test and debug code written in python
CO-2	To create applications using Python Programming
CO-3	To perform file operations to read and write data in files
CO-4	To write programs for general purpose I/O devices using MicroPython

Practical – Course Outcome matrix

Sr. No.	Objective(s) of Experiment	CO1	CO2	CO3	CO4
1.	Write a python programs using arithmetic and logical operators, bitwise operators, assignment operator	√			
2.	Write a python programs using decision making statements if, elif, else.	√			
3.	Write a python programs using control statements and loops.	√			
4.	Write a python programs using python string methods.		√		
5.	Write a python programs of list methods.		√		
6.	Write python programs to create functions and use functions in the program.		√		

7.	Write programs for generating different types of plots using Python.				√
8.	Write python programs of file handling operations.				√
9.	To become familiar with MicroPython and NodeMCU. Configure NodeMCU for MicroPython. Write Micropython test program to blink LED		√		√
10.	Connect Digital/Analog I/O module with NodeMCU and write program to read status of switch and display it on LED, Read and display temperature in MicroPython.		√	√	
11.	MicroPython program to display sensor data on web browser using WiFi connection (Node MCU as web server and PC/Desktop/Smartphone as a client)		√		√
12	Read analog value using NodeMCU and and write ADC reading in file using PC. Use WiFi for communication between NodeMCU and PC		√		√
13	Connect NodeMCU with with WiFi Access Point and transmit data from NodeMCU to Cloud. Connect Digital/Analog I/O module with NodeMCU and send temperature and light data on cloud (Thingspeak, Firebase, Ubidot or any other cloud service)				
14	Student Micro Project				

Instructions to the students:

Micro/Mini Project Report

It is compulsory for all students to create micro/mini project using embedded board (NodeMCU, Texas Tiva, RISCV or any other …) It should have problem statement, schematic diagram, program code and all relevant information of components used for the project.

Experiment No.1

Aim: Write a python programs using arithmetic and logical operators, bitwise operators, assignment operator.

Introduction:

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Advantages of Python:

- Easy to read and easy to learn
- Python focuses on code readability
- Python is free and open source
- Data science library and other extensive libraries
- Extensible
- Python is chosen by the best in the world, companies like Google, Facebook or Microsoft, and it's growing very fast
- Python is an obvious choice for artificial intelligence, machine learning, data analysis and visualization because allows faster development

Limitations of Python:

- Slower execution speed: Python is slow as compared to C/C++
- Weak for mobile application development
- Large memory consumption (So subset MicroPython is developed for embedded applications)
- Difficult to test : Since it is an interpreter based language, it is tough to run tests on code written in Python. All errors and bugs turn up only during the run time, which makes it very tough to test code snippets written in Python.
- Incompatibility of Two Versions Python2 and Python3

Popular applications of Python:

- Scientific computing
- Artificial intelligence and Machine learning
- Computer vision and automation
- Embedded Systems development
- Interne of Things
- Game development, Web development
- Web Server Programming (Python libraries like DJango and Flask)
- Data Science, Cryptography, Ethical hacking

Installation of Python:

- Install latest version of python 3.8 from the website https://www.python.org/
- After Installation of Python you can use python prompt in Command windows or you may use Python's IDLE (Integrated Development and Learning Environment)

You may use following alternatives of IDLE for interactive execution of python command.

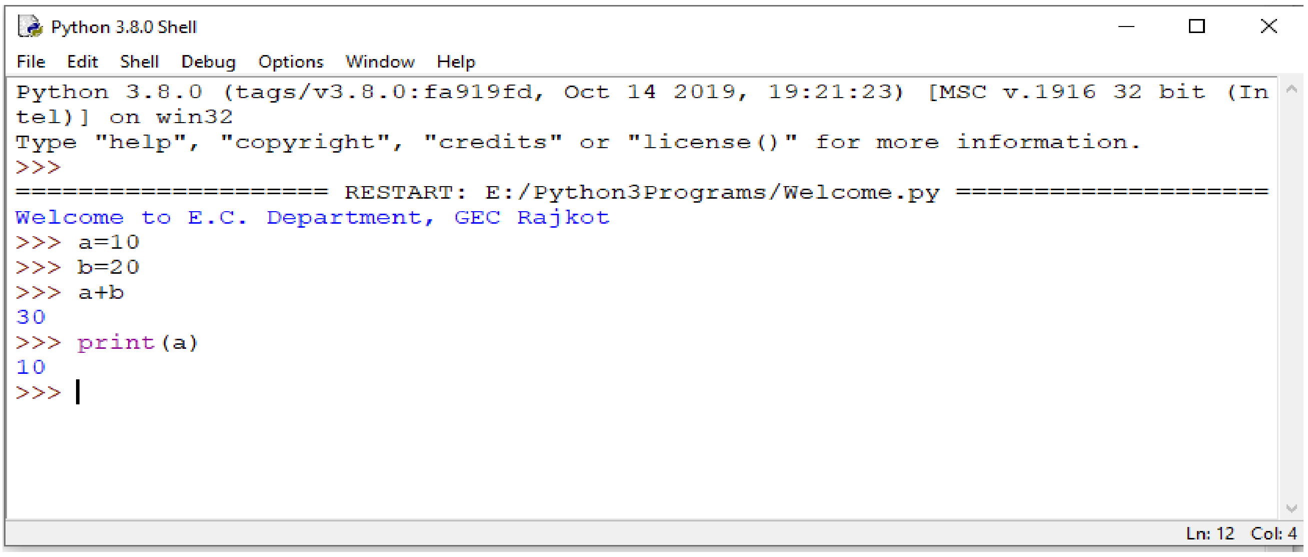
- Install Ipython (Interactive Python) can be installed using following command:

  $ pip install ipython (From command prompt)

- Alternately visit **https://www.anaconda.com/** and install anaconda (It has python installation along with required modules). Jupyter notebook will be automatically installed in anaconda provides interactive python terminal in anaconda distribution.
- We can also use **https://colab.research.google.com/** Online interactive python notebook of Google. Google Colab allows you to use Python notebook and GPU for computationally intensive task. It does not need installation of Python in local machine

Few Screen shots:

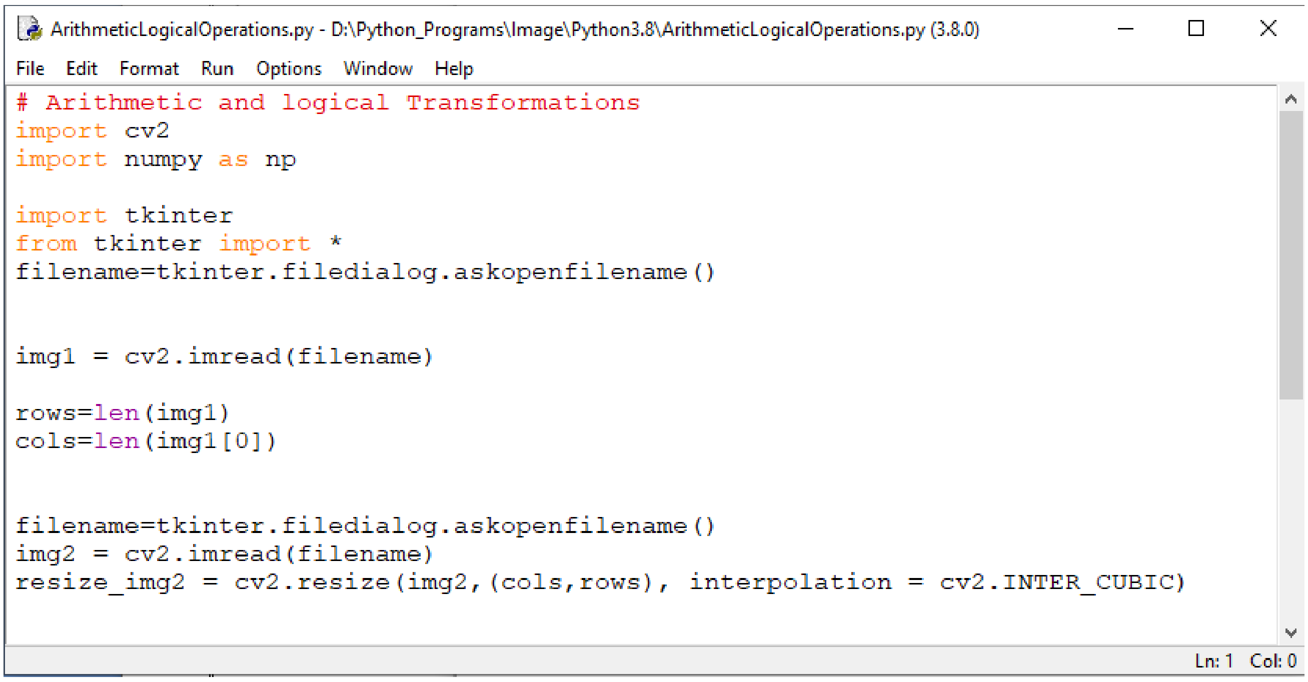
Python terminal (Shell or command prompt):



```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: E:/Python3Programs/Welcome.py ====================
Welcome to E.C. Department, GEC Rajkot
>>> a=10
>>> b=20
>>> a+b
30
>>> print(a)
10
>>> 
```

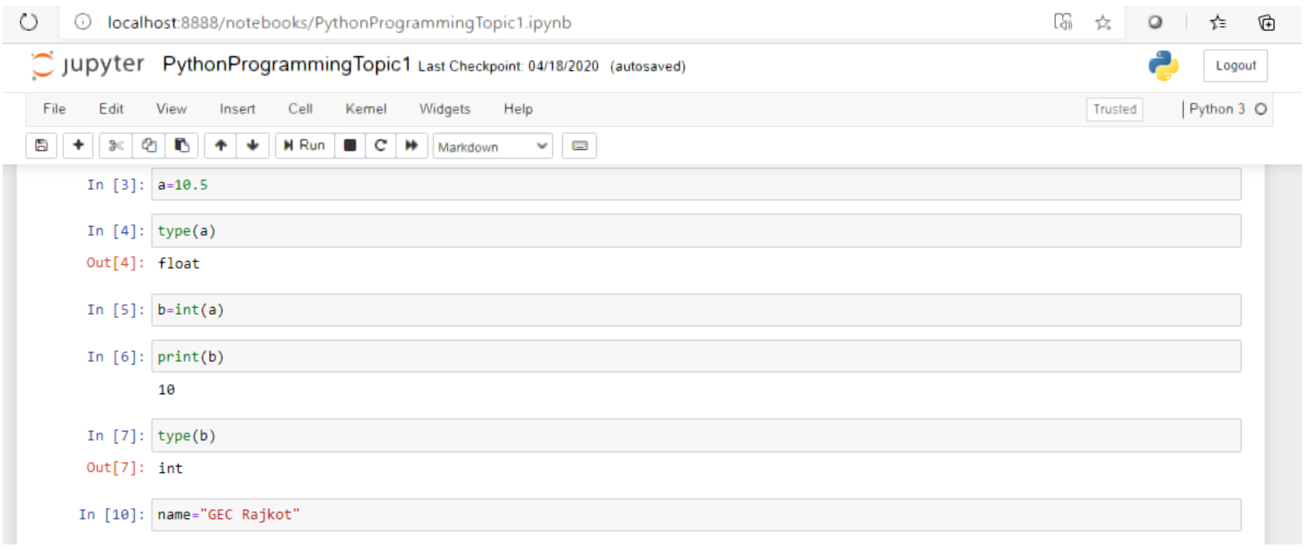
Python IDLE:

It provides facility to write program in file and execute it using Run module or F5 short cut.


ArithmeticLogicalOperations.py - D:\Python_Programs\Image\Python3.8\ArithmeticLogicalOperations.py (3.8.0)

```
# Arithmetic and logical Transformations
import cv2
import numpy as np

import tkinter
from tkinter import *
filename=tkinter.filedialog.askopenfilename()


img1 = cv2.imread(filename)

rows=len(img1)
cols=len(img1[0])


filename=tkinter.filedialog.askopenfilename()
img2 = cv2.imread(filename)
resize_img2 = cv2.resize(img2,(cols,rows), interpolation = cv2.INTER_CUBIC)
```

Jupyter notebook:

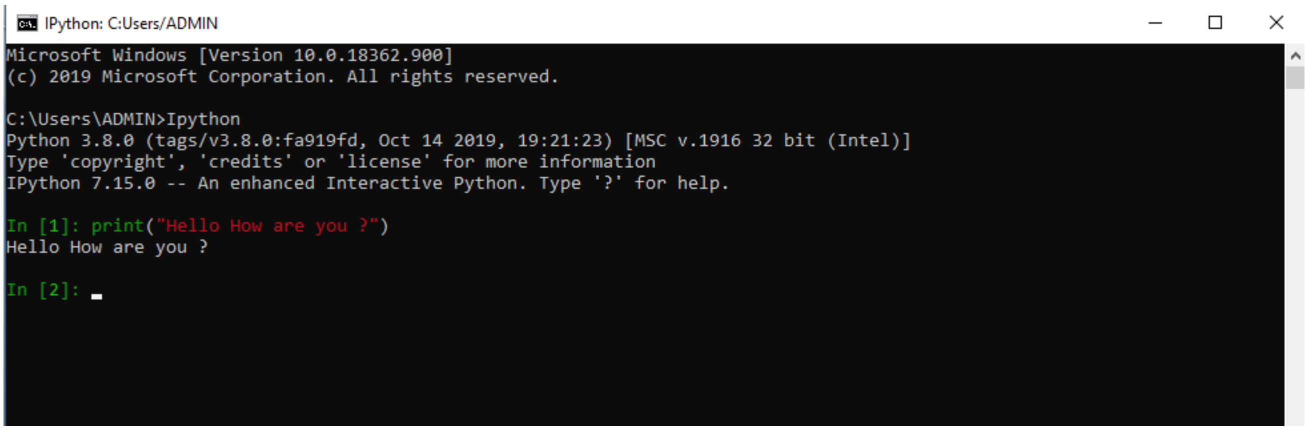

localhost:8888/notebooks/PythonProgrammingTopic1.ipynb

jupyter PythonProgrammingTopic1 Last Checkpoint: 04/18/2020 (autosaved)

```
In [3]: a=10.5

In [4]: type(a)
Out[4]: float

In [5]: b=int(a)

In [6]: print(b)
        10

In [7]: type(b)
Out[7]: int

In [10]: name="GEC Rajkot"
```

- Jupyter notebook as many facilities like it saves all work (all commands executed) automatically in Python notebook.
- Press CTL+Enter to execute command in the cell
- Press ALT+Enter to execute command and insert new cell

IPython interactive terminal



```
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>Ipython
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.15.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print("Hello How are you ?")
Hello How are you ?

In [2]:
```

Sample Python Program 1:

Execute following program in terminal:

```
>>> width=20
>>> height=5*9
>>> area=width*height
>>> print(area)
900
>>> radius=10
>>> area_circle=3.14159*(radius**2)
>>> print(area_circle)
314.159
>>> type(width)
<class 'int'>
>>> type(radius)
<class 'int'>
>>> type(area_circle)
<class 'float'>
>>> type(area)
<class 'int'>
```

Explanation:

Python does not require variable declaration. We can use variables directly and if we want to know type of variable, use type() function.

After executing above program write it (as shown following) in area.py file and run the program. You can use inbuilt editor of IDLE to write the program.

```
print("Welcome to E.C. Department, GEC Rajkot")
```

```
width=20
height=5*9
area=width*height
print("Area: ")
print(area)
radius=10
area_circle=3.14159*(radius**2)
print("Area of Circle: ")
print(area_circle)
type(width)
type(radius)
type(area_circle)
type(area)
```

Output of the program when run on IDLE or Jupyter:

```
Welcome to E.C. Department, GEC Rajkot

Area:

900

Area of Circle:

314.159
```

You can also try this program in interactive python or jupyter notebook. We have not used module numpy in this program.

Sample Program 2:

Write following program and save as ALOperations.py in your folder

```
import sys;
print("Arithmetic and Bitwise Logical Operations")
n1=int(sys.argv[1]);
n2=int(sys.argv[2]);
print("Addition of given numbers:")
print (n1+n2)
print("Multiplication of given numbers:")
print (n1*n2)
print("Division of first number by second number:")
print (n1/n2)
print("Bitwise Logical AND operation: ")
print(n1 & n2)
print("Bitwise Logical OR operation: ")
print(n1 | n2)
print("Bitwise Logical EX-OR operation: ")
print(n1 ^ n2)
```

- Open DOS SHELL (Command prompt) and go to location of your folder where python file is saved with CD (Change Directory) command

python ALOperations argument1 argument2 (It will also execute even if we do not write python)

For example issue following command in command prompt:

```
E:\Python3Programs>ALOperations 10 20
```

Output of the program:

```
Arithmetic and Bitwise Logical Operations
Addition of given numbers:
30
Multiplication of given numbers:
200
Division of first number by second number:
0
Bitwise Logical AND operation:
0
Bitwise Logical OR operation:
30
Bitwise Logical EX-OR operation:
30
```

In this program we use module sys and imported in the program. It has function argv which can be called by sys.argv() to accept arguments from the use from command line.

Now start writing different python programs using different operators and write in worksheet.

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Modulus %
- Exponential **
- Assignment =
- Comparison ==, != (<>), <, >, <=, >=
- Bitwise AND: &
- Bitwise OR: |
- Bitwise EX-OR: ^

• Bitwise Complement:	~
• Bitwise shift left:	<<
• Bitwise shift right:	>>
• Logical AND:	and
• Logical OR	or
• Logical NOT	not

Write different programs using above operators.

:: WORKSHEET ::

[1] Execute following program and write output:

```
import numpy as np
n1=int(input("Enter value of n1: "));
n2=int(input("Enter value of n2: "));
n3=n1<<1
n4=n2>>4
n5=n1%n2
n6=~n1
n7=n1^n2
n8=np.pi*(n1**2)
print(n1)
print(n2)
print(n3)
print(n4)
print(n5)
print(n6)
print(n7)
print(n8)
```

Output of the program:

[2] Execute the program for Arithmetic Operator.

- Multiply 10 with 5, and print the result.
- Add 20 with 66, and print the result.

- Subtract 88 with 52, and print the result.
- Divide 20 with 6, and print the result.
- Modulo 52 with 4, and print the result.
- Exponantion 5 with 66, and print the result.
- Floor division of 20 with 6, and print the result.

(3) What will be the output of following expression in python?

- 4 + 5.2 – 0.2
- 9.8 – 2.1 * 3
- 2 * 3 – 4 // 2
- (10 + 4 * 2) / 9 + 1
- (True or False) and (not (True and False))
- (2**3)%2==4
- ((2*5)+4)-8%3**2
- 2.5/0.5/2
- 2.5/0.5//2
- 5**3+3**4>=200

(4) Comparision operarator

- Write a program to use comparison operator to find the relation between given two numbers (user input number).
- (Greater than, Less than, Equal to, not equal to, Greater than or equal to, Less than or equal to).

(5) Logical operator

- If A=True and B=False than print the result of and, or, not of given operands.

(6) Assignment operators

- For A=66 and B=8, use all the different assignment operator and print the result.

(7) Membership Operator

- 'hello' in 'hello world'
- 'name' in 'hello world'
- b=[1,2,3,4,5] 5 not in b

Experiment No. 2

Aim: Write a python programs using decision making statements if, elif, else.

Objectives:

- To get familiarity with basic step to run control statements of python program.

Core Concept

Python conditional statements are constructs that allow a program to make decisions and alter its execution flow based on specific conditions. They introduce branching logic into the code, enabling different code paths to be executed depending on whether a given condition is true or false.

Fundamental Constructs

- **if statement:** Executes a block of code if a condition evaluates to True.

  Python

  ```
  if condition:
     # Code to be executed if condition is True
  ```

- **else statement:** Provides an alternative block of code to be executed if the if condition is False.

if condition: # Code to be executed if condition is True else: # Code to be executed if condition is False

* **`elif` statement:**

 Allows for multiple conditions to be checked sequentially.
```python

```
if condition1:
   # Code to be executed if condition1 is True
elif condition2:
   # Code to be executed if condition2 is True
else:
   # Code to be executed if

 none of the conditions are True
```

How They Work

1. **Condition Evaluation:** The condition within the if, elif, or else statement is evaluated to a Boolean value (True or False).
```

2. **Code Execution:** If the condition is True, the code block associated with that statement is executed. If False, the program proceeds to the next elif or else block, if present.
3. **Indentation:** Python relies on indentation to define code blocks. The code within a block must have the same indentation level.

Underlying Logic

Conditional statements implement a decision-making process. The program's control flow is diverted based on the outcome of the condition. This branching capability is essential for creating dynamic and responsive programs.

Importance

Conditional statements are fundamental to programming and are used extensively in various applications:

- **Controlling program flow:** Determining which code sections to execute based on input or data.
- **Error handling:** Implementing error checks and recovery mechanisms.
- **Decision-making:** Simulating real-world decision-making scenarios.
- **Data processing:** Filtering and manipulating data based on specific criteria.

Key Points

- Conditions are expressions that evaluate to Boolean values.
- Indentation is crucial for defining code blocks.
- Multiple conditions can be checked using elif.
- The else block is optional and provides a default path.

By understanding the theory behind Python conditional statements, you can effectively use them to create well-structured, efficient, and flexible programs.

Exercise

1) Write a program to check the given number is Odd or Even.
2) Write a program to check if a given number is positive, negative or zero.
3) Write a program to swap the two numbers without using temporary variable.
4) Write a program to find Largest among given Three Numbers.
5) Write a program to sum of ODD and EVEN numbers from 1 to 10.
6) Write a program make a simple calculator that can add, subtract, multiply and divide using user choice.
7) Write a python program that checks a person's age and prints an appropriate message. (Minor, Adult, Senior citizen, Invalid)
8) Write a program to calculate SPI using given marks.

Experiment No. 3

Aim: Write a python programs using control statements and loops.

Objectives:

- To get familiarity with basic step to run for loop and if else statements of python program.

Core Concept

Loops in Python are control flow statements that allow you to execute a block of code repeatedly. They are essential for automating repetitive tasks and iterating over collections of data. Python primarily offers two types of loops: `for` and `while`.

The `for` Loop

- **Iterates over a sequence:** The `for` loop is used to iterate over elements in a sequence (like lists, tuples, strings, or other iterable objects).
- **Syntax:**

  Python

  ```
  for item in sequence:
     # code to be executed for each item
  ```

- **Mechanism:**
  1. The loop initializes an iterator for the sequence.
  2. In each iteration, the next item from the sequence is assigned to the loop variable (e.g., `item`).
  3. The code block is executed with the current item.
  4. The loop continues until all items in the sequence have been processed.

The `while` Loop

- **Repeats based on a condition:** The `while` loop continues to execute as long as a given condition is True.
- **Syntax:**

  Python

  ```
  while condition:
     # code to be executed while the condition is True
  ```

- **Mechanism:**
  1. The condition is evaluated.

2. If the condition is True, the code block is executed.
3. The process repeats from step 1.
4. The loop terminates when the condition becomes False.

Loop Control Statements

- **break:** Exits the loop prematurely.
- **continue:** Skips the current iteration and moves to the next.
- **pass:** Does nothing, often used as a placeholder.

Importance of Loops

- **Efficiency:** Automate repetitive tasks.
- **Data processing:** Iterate over data structures.
- **Algorithm implementation:** Create iterative algorithms.
- **Control flow:** Modify program execution based on conditions.

Key Points

- Indentation is crucial for defining the loop body.
- Infinite loops can occur if the while condition never becomes False.
- The for loop is often preferred for iterating over sequences.
- The while loop is more flexible for custom iteration conditions.

Exercise

1) Write a Python program to find out the sum of all digits of a number.
2) Write a Python program to print all combinations of three numbers.
3) Write a Python program to calculate total vowels in a string.
4) Python program to find the smallest and largest divisor of a number.
5) Write a python program to reverse a number.
6) Write a python program to check if a number is abundant/excessive or not.
7) Write a python program to check if a year is a leap year or not.
8) Write a python program to display all the prime numbers within an interval range given by user.
9) Write a Python program to find the LCM of two numbers.
10) Write a program to print the following pattern in output console window.

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

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

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

```
*
```

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

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

Experiment No. 4

Aim: Write a python programs using python string methods.

Objectives:

- Understand python string methods and characteristics

Introduction:

A string in Python is a sequence of characters. It is an immutable data type, meaning once created, its value cannot be changed. Strings are enclosed within either single quotes ('') or double quotes ("").

Key Characteristics of Strings:

- **Sequence:** Strings are ordered collections of characters.
- **Immutable:** Once created, the contents of a string cannot be modified.
- **Indexing:** Each character in a string has an index, starting from 0.
- **Slicing:** You can extract substrings using slicing.
- **Iterability:** You can iterate over characters in a string using loops.
- **Rich Methods:** Python provides a rich set of built-in methods for string manipulation.

String Representation in Memory

Python stores strings in an efficient manner. Internally, strings are represented as an array of characters. To save memory, Python often uses a technique called "string interning," where identical strings point to the same memory location.

String Operations

- **Concatenation:** Combining two or more strings using the + operator.
- **Replication:** Creating multiple copies of a string using the * operator.
- **Membership Testing:** Checking if a substring exists within a string using in and not in operators.
- **Indexing and Slicing:** Accessing individual characters or substrings using indices and the slicing operator [start:end:step].
- **String Methods:** Python provides numerous built-in methods for string manipulation, such as upper(), lower(), strip(), split(), join(), find(), replace(), and many more.

String Immutability

Although strings are immutable, new strings can be created by modifying existing ones. For example, new_string = old_string.upper() creates a new string in uppercase, leaving the original string unchanged.

Strings are widely used in Python for:

- Text processing
- Data manipulation
- User input and output
- File handling
- Creating formatted output
- Building web applications

Exercise

1. Write a program to concatenate two strings.
2. Write a program to reverse a given string.
3. Write a program to convert a string to uppercase and lowercase.
4. Write a program to split a string into a list of substrings based on a delimiter.
5. Write a program to join a list of strings into a single string with a specified delimiter.
6. Write a program to count the occurrences of a substring in a string.
7. Write a program to format strings using the format() method.
8. Write a program to align strings (left, right, center) using formatting methods.
9. Write a program to check if a given string is a palindrome.
10. Write a program to check if two strings are anagrams.
11. Write a program to count the number of vowels and consonants in a string.
12. Write a program to remove duplicate characters from a string.
13. Write a program to reverse the words in a given sentence.

Experiment No. 5

Aim: Write a python programs of list

Objectives:

- Understand Python list methods

Introduction

A Python list is a versatile, ordered, and mutable collection of items. It can hold elements of various data types, including integers, floats, strings, and even other lists. Lists are defined by enclosing elements within square brackets [ ] and separated by commas.

Key Characteristics of Lists:

- **Ordered:** Elements have a specific order, and their position can be accessed using an index.
- **Mutable:** Elements can be changed, added, or removed after list creation.
- **Dynamic:** Lists can grow or shrink in size as needed.
- **Heterogeneous:** Lists can contain elements of different data types.

How Lists Work Internally

Python lists are implemented as dynamic arrays. This means that they initially allocate a certain amount of memory to store elements. As more elements are added, the list might need to reallocate memory to accommodate the growth. This process can be efficient for most operations, but it can lead to performance implications for frequent insertions or deletions at the beginning of the list.

List Operations

- **Accessing Elements:** Use indexing (starting from 0) to access individual elements.
- **Slicing:** Extract a portion of the list using slicing syntax (e.g., list[start:end:step]).
- **Modifying Elements:** Assign new values to elements using their index.
- **Adding Elements:** Use append() to add elements to the end, insert() to insert at a specific position, or extend with another list using extend().

- **Removing Elements:** Use remove() to remove by value, pop() to remove by index, or del to delete a range of elements.
- **List Comprehension:** Create new lists based on existing lists using concise syntax.

List Methods

Python provides a rich set of built-in methods for list manipulation, including:
count(): Count the number of occurrences of an element.
index(): Find the index of the first occurrence of an element.
sort(): Sort the list in ascending order.
reverse(): Reverse the order of elements.
`copy()`: Create a shallow copy of the list.
`clear()`: Remove all elements from the list.

Exercise

1) Write a python program to average all numbers from the given list. (Assume that List contain string and integer)
2) Write a python program to delete a specific element (all occurrence) from the user list. (Create a list from user input).
3) Write a python program to delete all duplicate elements from a list.
4) Write a python program to find intersection and union of two list elements.
5) Write a Python program to flatten a nested list or list of lists.
6) Write a Python program to find the repeated items of a tuple.
7) Write a Python program to convert a list to a tuple.
8) Write a Python program to convert a tuple to a dictionary.
9) Write a Python program to find the index of an item of a tuple.
10) Write a Python program to reverse a tuple.

Experiment No. 6

Aim: Write python programs to create functions and use functions in the program.

Objectives:

- The aim of this experiment is to understand the concepts of functions in Python, including defining functions, calling functions, passing arguments, and returning values.

Introduction

A function in Python is a block of organized, reusable code that is used to perform a specific task. It provides better modularity for your application and a high degree of code reusing.

Key Components of a Function:

- **Function Definition:**
    - Uses the def keyword followed by the function name.
    - Parameters (optional) are defined within parentheses.
    - The function body is indented.
    - An optional return statement specifies the output.
- **Function Call:**
    - Invokes the function by its name, passing arguments if necessary.

Parameters and Arguments:

- **Parameters:** Variables defined in the function definition to receive values from the caller.
- **Arguments:** Actual values passed to the function when it is called.

Return Value:

- A function can optionally return a value using the return statement.
- If no return statement is present, the function returns None.

Scope and Lifetime:

- **Scope:** The region of code where a variable can be accessed.
- **Lifetime:** The duration for which a variable exists.
- Variables defined within a function have local scope and lifetime.

- Variables defined outside functions have global scope and lifetime.

Types of Functions:

- **Built-in Functions:** Predefined functions provided by Python (e.g., print, len, type).
- **User-defined Functions:** Functions created by the programmer.
- **Recursive Functions:** Functions that call themselves directly or indirectly.
- **Lambda Functions:** Anonymous functions created using the lambda keyword.

Exercise

1. Write a python function that finds the decimal equivalent of the binary number 10011?
2. Write a python function that finds the equivalent Hexa-decimal code and octal code for the decimal number 191.
3. Write a python function isIn that accepts two strings as arguments and returns True if either string occurs anywhere in the other, and False otherwise.
4. Create a Python function to take user input and check validity of a password.
5. Create a python function that sort all words of a user String in Alphabetical order.
6. Create a Python function that check if a enter date is valid or not.
7. Create a Python function that find the number of days between two dates.
8. Write a python program to find the factorial of a given number using recursive function.
9. Write a python program to find the Fibonacci series for given number using recursive function.
10. Write a Python program to find the even integers from the given list using Lambda.
11. Write a Python program to find the square values and cube values of every integer of given list using Lambda.
12. Write a Python program to find the palindrome strings if any in a given list using Lambda.

Experiment No. 7

Aim: Write programs for generating different types of plots using Python.

Objectives:

- The aim of this experiment is to understand how to create various types of plots using Python's plotting libraries such as Matplotlib and Seaborn. This includes creating line plots, bar plots, histograms, scatter plots, and box plots.

Introduction

Matplotlib is a versatile plotting library in Python, ideal for creating static, animated, and interactive visualizations. Its main goal is to equip users with the tools to graphically represent data, simplifying analysis and comprehension. Originally developed by John D. Hunter in 2003, Matplotlib is now maintained by a robust community of developers.

Matplotlib boasts several key features that make it a powerful plotting library. Its versatility allows for the generation of diverse plots, including line, scatter, bar, histograms, and pie charts. The library offers extensive customization options, enabling control over plot elements such as styles, colors, markers, and labels.

Matplotlib produces high-quality, publication-ready plots with detailed aesthetic control, making it ideal for professional presentations. It integrates seamlessly with NumPy, allowing for direct plotting of data arrays. Additionally, Matplotlib supports dynamic data exploration through interactive plots and widgets.

Its cross-platform capability ensures it runs smoothly on various operating systems, including Windows, macOS, and Linux. The library's extensibility is another strength, as it supports numerous add-ons and extensions, such as Seaborn, Pandas plotting functions, and Basemap for geographical plotting.

Basic Components of a Matplotlib Figure

1. **Figure**: The top-level container that serves as the canvas for the plot, like a blank sheet of paper.
2. **Axes**: Rectangular areas within the figure where data is plotted; a figure can contain multiple axes arranged in rows and columns.
3. **Axis**: Represents the x-axis and y-axis, defining data limits, tick locations, and labels.
4. **Markers**: Symbols (e.g., circles, squares, triangles) used to denote individual data points, often in scatter plots.

5. **Lines**: Connect data points in plots (e.g., line plots) and can be styled in various ways to show trends or relationships.
6. **Title**: Provides a descriptive heading for the plot, usually at the top of the figure.
7. **Axis Labels**: Text elements that describe the data on the x-axis and y-axis, providing context and units.
8. **Ticks**: Small marks along the axis that indicate specific data points or intervals.
9. **Tick Labels**: Text elements that provide labels for the tick marks, showing data values corresponding to each tick.
10. **Legend**: Offers a key to the symbols or colors representing different data series or categories.
11. **Grid Lines**: Horizontal and vertical lines extending across the plot, corresponding to specific data intervals or divisions.
12. **Spines**: Lines that form the borders of the plot area, separating it from the surrounding whitespace and customizable to change the plot's appearance.

Different Types of Plots in Matplotlib

Matplotlib offers a wide range of plot types to suit various data visualization needs. Here are some of the most commonly used types of plots in Matplotlib:

- Line Graph
- Stem Plot
- Bar chart
- Histograms
- Scatter Plot
- Stack Plot
- Box Plot
- Pie Chart
- Error Plot
- Violin Plot
- 3D Plots

Exercise

1. Write a python program to create a line plot on the basis of selected database.
2. Write a python program to create a pie plot on the basis of selected database.
3. Write a python program to create a bar and barh plot on the basis of selected database.
4. Write a python program to create a histogram plot on the basis of selected database.
5. Write a python program to create a scatter plot on the basis of selected database.
6. Write a python program to create a subplot contain four different line plot from the database.

Select the database (CSV or EXCEL) of your own choice / interest.

Experiment No. 8

Aim: Write python programs of file handling operations.

Objectives:

Understanding fundamental concepts of file handling operation in Python

Introduction

A file is a named location on a storage device used to store related information. Python treats files as sequences of characters or bytes. File handling is essential for tasks such as data storage, retrieval, and manipulation. It allows Python programs to interact with external files, enabling data persistence, configuration management, logging, and more complex operations like data analysis and processing.

Core Concepts

- **File Object:** A representation of a file in memory created by the open() function.
- **File Modes:** Specify how the file will be used (read, write, append, etc.).
- **File Access:** Reading, writing, or appending data to a file.
- **File Closing:** Releasing system resources associated with the file.

Python provides built-in functions for creating, writing, and reading files. Two types of files can be handled in Python, normal text files and binary files (written in binary language, 0s, and 1s).

Text files: In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in Python by default.

Binary files: In this type of file, there is no terminator for a line, and the data is stored after converting it into machine-understandable binary language.

Opening a File

The open() function is used to open a file. It returns a file object.

File Modes

- **'r'**: Read mode (default).
- **'w'**: Write mode (creates a new file or overwrites an existing one).
- **'a'**: Append mode (adds to the end of an existing file or creates a new one).
- **'x'**: Create mode (creates a new file, raises an error if it already exists).

- **'b'**: Binary mode (for non-text files).
- **'+'**: Open for both reading and writing.

Reading from a File

- read(): Reads the entire contents of the file as a string.
- readline(): Reads a single line from the file.
- readlines(): Reads all lines of the file as a list of strings.

Writing to a File

- write(): Writes a string to the file.
- writelines(): Writes a list of strings to the file.

Closing a File

The close() method is used to close a file. It's essential to close files to release system resources.

Exercise

1. Write a python program to create a new file.
2. Write a python program to read a file.
3. Write a Python program to append text to a file and display the text.
4. Write a Python program to read a random line from a file.
5. Write a python program to count the number of words in a file.
6. Write a python program to count the number of blank spaces in a file.
7. Write a python program to count the total number of lines in a file.
8. Write a python program to find the longest words.
9. Write a Python program to count the frequency of words in a file.
10. Write a Python program to copy the contents of a file to another file.
11. Create a program to write on to file (Your Home Address) and read the same file and display its content on console.
12. Write a program to open any non-text file saved in your computer and read that file.
13. Write a program to create a CSV file of following details (at least 5 Data) and the same file and display its content on console.

Sr. No.	First Name	Middle name	Surname	DOB	M/F	Contact number	Email-id

Experiment No. 9

Aim: To become familiar with MicroPython and NodeMCU. Configure NodeMCU for MicroPython. Write Micropython test program to blink LED

Objectives:

- Understand MicroPython programming
- Upload sample micropython program to NodeMCU to blink LED

Introduction:

Micropython allows us to run miniature version of Python 3 on embedded boards and microcontrollers.

- ❖ MicroPython is tiny open source Python programming language interpreter designed to run on small embedded devices.
- ❖ MicroPython does not support entire python standard library so we can call it as sub-set of python.

Advantages:

- ❖ Small foot print (Requires less memory)
- ❖ Near to hardware due to Hardware API
- ❖ Clean and simple Python code to control hardware

Disadvantages:

- ❖ Limited standard library functions
- ❖ Execution speed less than C so sometimes it may create problem for real time signal processing where execution speed requirement is critical

There are two methods to install micropython in NodeMCU.

Use any one of following method to upload firmware of micropython to your NodeMCU.

[1] Use ESP8266 flasher (nodemcuflahser Available in Advance Microprocessor folder)

Select binary file esp8266-20170823-v1.9.2.bin (or latest version)

Click on Flash

[2] Using esptool in command prompt

- Open command prompt (cmd)
- Install esptool

  (If not installed then) Command: >pip install esptool

- To erase esp8266:

  ```
  >esptool.py  -- port COM6 erase_flash
  ```

- Download firmware in esp8266 for micropython

  ```
  > esptool.py --port COM6 --baud 460800 write_flash
          -- flash_size=32m 0 esp8266-20170823-v1.9.2.bin
  ```

  In above command give path of binary file for the firmware if it is not there in the same directory, use flash_size=detect if ROM size is unknown (Above command can be written as BATCH file.

Sample Program To Blink LED of NodeMCU

LED is connected with GPIO2 (D4 Pin) in NodeMCU board. Open ESPLorer to write program for LED blinking

- Start ESPLorer. Go to Settings and select Micropython
- Write program in micropython (Script)
- Connect NodeMCU and select corresponding COM port , Set baud rate 115200
- Open the port and press reset button on NodeMCU, Command prompt will display
- Open new file and write program
- Click on “Send to ESP”.

Micropython Program to blink LED:

```
import machine              #import machine module from firmware
import time                 #import time module from firmware

LED = machine.Pin(2, machine.Pin.OUT)  #Initialize GPIO2 as output
```

```
#do infinetely
while True:
   LED.on()          #pin status ON
   time.sleep(0.1)     #time delay of 0.1 second
   LED.off()          #pin status OFF
   time.sleep(0.1)     #time delay of 0.1 second
```

WORKSHEET

[1] Modify sample code to blink internal LED of NodeMCU with different delay.

[2] Connect external LED with help of series resistor 470 Ω at pin D4 (GPIO2) and modify sample program to blink both LEDs (Internal as well as external) alternately at the interval of 1 second. Use bread board for the connection. Write modified program.

Experiment No. 10

Aim: Connect Digital/Analog I/O module with NodeMCU and write program to read status of switch and display it on LED, Read and display temperature in MicroPython.

Objectives:

- MicroPython program to read switch and display status on LED
- MicroPython program to read temperature using LM35 and display at command prompt.

Sample program: To read switch and display its status

Write program to read status of switch (digital sensor) connected at pin D1 of NodeMCU (GPIO5), Make internal LED ON if switch is pressed and OFF if switch is not pressed. (Use same steps of Program 1 to write and upload program using ESPlorer)

Micropython Program to read status of switch and reflect it on LED:

```
from machine import Pin    #import pin module from machine from
import time                 #import time module from firmware

digitalSensor = Pin(1, Pin.IN, Pin.PULL_UP) ##Initialize GPIO2 as D4 as input mode
LED=Pin(2,Pin.OUT)
#do infinitely
while True:
   status = digitalSensor.value()    #read the sensor value from the pin and store it
   if status==0:
      LED.off()
   else:
      LED.on()#print the value of sensor value on serial monitor
   time.sleep(1)              #provide time delay of 1 second
```

After uploading program in NodeMCU, connect switch of DAIO module to pin D1 of NodeMCU.

Sample Program: Read Temperature from DAIO module and display it on console.

Connect temperature sensor output (LM35) of DAIO Module with NodeMCU. Temperature will be displayed on console. If temperature is greater than 30, LED will remain OFF and if it is less than 30, LED will turn ON.

Micropython Program to read and display temperature:

```
import machine
import time
LED = machine.Pin(2,machine.Pin.OUT)
while True:
  sensor=machine.ADC(0)
  sum=0
  for i in range(0,100):
    sensordata=sensor.read()
    sum=sum+sensordata
    time.sleep_ms(1)
  sum=sum/100
  temperature=(sum*3300)/(1023*10);
  if temperature>30:
    LED.off()
  else:
    LED.on()
  print(temperature)
  time.sleep(5)
```

WORKSHEET

[1] Modify sample code given above to display light intensity on the console. Use LDR of DAIO board for this purpose. If Light intensity value>800, LED should become ON

[2] Modify above program to read light intensity and display “DARK” and “BRIGHT” depending on light intensity falls on LDR. Glow LED of NodeMCU in case of DARK condition.

[3] Interface Digital Humidity Temperature sensor DHT-22 or DHT-11 with NodeMCU and run following program.

```
import dht
import machine
import time
d = dht.DHT22(machine.Pin(4))
d.measure()
while True:
    time.sleep(5)
    print("Measuring.")
    d.measure()
    print("Temperature: %3.1f °C" % d.temperature())
    print("   Humidity: %3.2f %% RH" % d.humidity())
```

Write output of the program in following space:

[4] Run live micropython on http://micropython.org/live/ execute following program

```
# turn on an LED and print text
import pyb
i=0
while i<5:
        pyb.LED(1).on()
        pyb.delay(1000)
        pyb.LED(1).off()
        pyb.delay(1000)
        print('Count: ')
        print(i)
        i=i+1
print("End of the Program .... Good Buy ...");
```

Run following sample programs given on live simulator and understand program.

- LCD Display
- DAC –ADC Program
- Servo Motor

Experiment No. 11

Aim: MicroPython program to display sensor data on web browser using WiFi connection

Objectives:

- MicroPython program to read sensor data
- MicroPython program to transmit sensor data to client as a server

Introduction:

In this experiment we will configure NodeMCU as webserver. It will read data from analog sensor and transmit it to webpage.

Program: Server program running at NodeMCU will send analog data to PC/Laptop and client side web browser will display data

```
# EC Department, L D College of Engineering, Ahmedabad
# Send sensor data (ADC Value) to webpage
# Note down address and type that address on webbrowser
# For example: 192.168.1.102:80 Analog values will be displayed

from machine import ADC
import socket
import network
import time

#wifi as a stataion
def wifi_connect():
    import network
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print('connecting to network...')
        sta_if.active(True)
        sta_if.connect('INDIA', 'apec2004')  #Id and PWD
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())
    print("Yes ... Now connected to the network")

wifi_connect()

time.sleep(2)
LSB = 3.3/1024 # LSB for ADC
adc = ADC(0) # ADC object
addr = ('192.168.1.102',80)      # Replace this IP as per network connection
```

```
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind(addr)
serverSocket.listen(3)
while True:
    cl, addr = serverSocket.accept()
    cl_file = cl.makefile('rwb', 0)
    while True:
        line = cl_file.readline()
        if not line or line == b'\r\n':
            break
    Dn = adc.read()
    vin = LSB * Dn
    sVin = "{:.3f}".format(vin)
    # Prepare the response
    response = 'HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n'
    response += """<!DOCTYPE html>
    <html>
    <body>
    <h3>Voltage on pin A0 = %sV </h3>
    </body>
    </html>\n""" % sVin
    # Send the response to the client
    cl.send(response)
    cl.close()
```

Run above program at NodeMCU side and then open web browser in PC or mobile. Open webpage on URL: 192.168.1.102:80 (Sensor values will be displayed)

Connect Digital Analog I/O Module POT-1 pin with A0 pin. Change POT value and see changes in analog display.

Program: Server program running at NodeMCU will sense digital sensors and send status to PC/Laptop and client side web browser will display status

```
# EC Department, L D College of Engineering, Ahmedabad
# Send Digital pin status (D1,D2 and D3) to webpage
# D1: GPIO5   D2: GPIO4   and D3: GPIO0
# For example: 192.168.1.102:80 Digital Status of above pins will be displayed


from machine import Pin
import socket
import network

#wifi as a stataion
def wifi_connect():
    import network
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print('connecting to network...')
        sta_if.active(True)
        sta_if.connect('INDIA', 'apec2004')   #ssid=JJP, Password=12345678
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())
    print("Yes ... NodeMCU is connected to Network ")

wifi_connect()

pinNum = [5,4,0]
pins = ['pin5','pin4','pin0']
state = [0,0,0]
for i in range(0,3):
    pins[i] = Pin(pinNum[i], Pin.IN, Pin.PULL_UP)

addr = ('192.168.1.101', 80)
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind(addr)
serverSocket.listen(3)

while True:
    print("hello")
    cl, addr = serverSocket.accept()
   # print(c1)
    cl_file = cl.makefile('rwb', 0)
    while True:
        line = cl_file.readline()
        if not line or line == b'\r\n':
            break
    sInp = ''
    for i in range(0,3):
        state[i] = pins[i].value()
        sInp += str(state[i])

    # Prepare the response
    print("response sent")
    response = 'HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n'
    response += """<!DOCTYPE html>
    <html>
    <body>
    <h3>State of ESP8266 Pins [5:4:0] = [%s] </h3>
    </body>
    </html>\n""" % sInp
    # Send the response to the client
    cl.send(response)
    cl.close()
```

Run above program at NodeMCU side and then open web browser in PC or mobile. Open webpage on URL: 192.168.1.102:80 (Digital Sensor values will be displayed)

Output on web browser: Digital Sensor Data [5:4:0] = [111]

Connect your mobile phone with WiFi and check same with browser available in mobile phone.

Connect Digital Analog I/O board switches with NodeMCU GPIO pins 5 and 4. Change status of swtch and verifiy whether it is reflected on web browser or not.

WORKSHEET

[1] Consider that we want to display status of door (Whether it is open or close) Sensor gives data 1 when door is open and 0 when door is closed. Modify micro python program such that it displays “GREEN rectangle when door is open and RED rectangle when door is closed” in browser. Test your program with NodeMCU and DAIO Module. (Use internet to learn HTML coding for drawing of RED and GREEN rectangles)

[2] Write Micro-python program to develop GUI to display sensor data on web-browser

Experiment No. 12

Aim: Read analog value using NodeMCU and and write temperature reading in file using PC. Use WiFi for communication between NodeMCU and PC

Objectives:

- MicroPython program to read sensor data
- MicroPython program to transmit sensor data to client as a server

Introduction:

In this experiment we will configure NodeMCU as webserver. It will read data from analog sensor and transmit it to webpage.

Program at NodeMCU:

```
# Course: 3151108 Python Programming
# Experiment No. 10
# This Program run on NodeMCU side
# It sends ADC value to TCP client using TCPServer
from machine import ADC
import time
import network
import socket

#wifi as a stataion
def wifi_connect():
   import network
   sta_if = network.WLAN(network.STA_IF)
   if not sta_if.isconnected():
      print('connecting to network...')
```

```
        sta_if.active(True)
        sta_if.connect('INDIA', 'apec2004')   #ssid=JJP, Password=12345678
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())
    print("Yes ... Connected to Network")

wifi_connect()

LSB = 3.3/1024 # LSB for ADC
adc = ADC(0) # ADC object

# create a socket object
#print ("Hello")
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = '192.168.1.100'
port = 55000

# bind to the port
serverSocket.bind((host, port))
# queue up to 5 requests
serverSocket.listen(5)

while True:
    # establish a connection
    clientSocket,addr = serverSocket.accept()
    sRecv = clientSocket.recv(128).decode('utf8')
    print(sRecv)
    if sRecv == 'ADC0':
        Dn = adc.read()
        vin = LSB * Dn
        sVin = "{:.3f}".format(vin)
    else:
        sVin = 'Wrong parameter.'
    clientSocket.send(sVin)
    clientSocket.close()
```

Program at PC/Laptop:

```
import sys
import socket
import time
f = open("test.csv", "w")
f.write("Sensor data: \n")
count=0
if (len(sys.argv) == 2):
    while(count<10):
        adc = str(sys.argv[1])
        # create a socket object
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        host = "192.168.1.102"; # IP-address of the TCP server of NodeMCU
        port = 55000
        # connection to hostname on the port.
        s.connect((host, port))
        s.send(adc.encode('utf8')) # utf-8 coding is used
        sRecv = s.recv(128)
        s.close()
        sdata=sRecv.decode('utf8')
        print('Vin on A0, V: ' + sdata)
        f.write(sdata)
        f.write(',')
        time.sleep(3)
        count=count+1
f.close()
```

WORKSHEET

[1] Modify Python Program at PC/Laptop to save sensor data along with time stamp

Experiment No. 13

Aim: Connect NodeMCU with with WiFi Access Point and transmit data from NodeMCU to Cloud

Objectives:

- To understand how to connect NodeMCU with WiFi Access point
- To open account in Cloud services (For example ThingSpeak)
- Measure temperature at the interval of 10 second and transmit to Cloud

Introduction:

- NodeMCU is opensource IoT plateform connects to WiFi by inbuilt WiFi chip ESP8266. NodeMCU collects data and it can be transferred to cloud like ThingSpeak.
- ThingSpeak™ is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak

Sample Program 1: To connect NodeMCU with network

- We will import network module to configure WiFi connection. Give network SSID and password to connect NodeMCU with local access point. sta_if.ifconfig will return network settings like IP address, netmask, gateway and DNS

Micropython Program to blink LED:

```
# This program will connect NodeMCU with WiFi Access Point
def wifi_connect():
    import network
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print('Please wait … Connecting to network...')
        sta_if.active(True)
        sta_if.connect('ID', 'PWD')    #Example ssid=EC, Password=gecr@123
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())
wifi_connect()
```

Note down network configuration:

IP addresss: Netmask:

Gateway: DNS:

Sample program 2: To upload temperature data on ThingSpeak.com

- For this experiment, open your account in ThingSpeak.com, get write API (you will need write API in this program)
- We will import socket module in this program. TCP socket provide a reliable stream of bytes between the connected network devices
- Command ai = _socket.getaddrinfo(HOST, 443) returns IP address of HOST i.e. api.thingspeak.com. Using this IP address, we can make a socket and connect to server to upload temperature data.
- Some technical terms as per ThingSpeak API documentation

❖ **Channel** – The name for where data can be inserted or retrieved within the ThingSpeak API, identified by a numerical Channel ID
❖ **Field** – One of eight specific locations for data inside of a channel, identified by a number between 1 to 8 – A field can store numeric data from sensors or alphanumeric strings from serial devices or RFID readers
❖ **Status** – A short status message to augment the data stored in a channel
❖ **Location** – The latitude, longitude, and elevation of where data is being sent from
❖ **Feed** – The collective name for the data stored inside a channel, which may be any combination of field data, status updates, and location info
❖ **Write API Key** – A 16 digit code that allows an application to write data to a channel
❖ **Read API Key** – A 16 digit code that allows an application to read the data stored in a channel

Micropython Program to read temperature and upload on ThingSpeak.com:

```
import usocket as _socket          #import usocket as _socket

from machine import ADC            #import ADC module from machine module from
firmware
import ussl as ssl                 #import ssl module from firmware
import network                     #import network module from firmware
import time                        #import time module from firmware

#function for identify wifi
nic = network.WLAN(network.STA_IF)
nic.active(True)
nic.connect('SSID', 'password')

temp = 0
```

```
#API_KEY for Write & HOST as www.thingspeak.com
API_KEY = "Your Write API Key "      # Example: "D8YB8F0FTYZ75QW8"
HOST = "api.thingspeak.com"



#takes the value from sensor and send it to HOST
while True:

    #sense the Data from the sensor
    adc = ADC(0)
    temp = adc.read()
    temp = (temp*3300)/(1023*10);
    #Send the data to the HOST
    data = b"api_key="+ API_KEY + "&field1=" + str(temp)
    s = _socket.socket()
    ai = _socket.getaddrinfo(HOST, 443)       #Connecting to the HOST
    addr = ai[0][-1]
    s.connect(addr)
    s = ssl.wrap_socket(s)
    s.write("POST /update HTTP/1.0\r\n")
    s.write("Host: " + HOST + "\r\n")
    s.write("Content-Length: " + str(len(data)) + "\r\n\r\n")
    s.write(data)
    print(s.read(128))
    s.close()
    time.sleep(1)
```

WORKSHEET

[1] Note down following parameters from your ThingSpeak account

Channel ID:

Write API Key:

Read API Key:

Add another field Light and upload light intensity data on ThingSpeak. Export data from ThingSpeak account.

[2] Copy and paste graphs plotted on ThingSpeak cloud for temperature and light in following space.

[3] Find out details about MQTT protocol and write here:

Experiment No. 14

Micro/Mini Project Report

It should have problem statement, schematic diagram, program code and all relevant information of components used for the project.