**GOVERNMENT ENGINEERING COLLEGE,**

**GANDHINAGAR**



**A Laboratory Manual for**

Python Programming (3151108)

B.E. Semester 5

(Electronics and Communication Engineering Department)



**Directorate of Technical Education, Gandhinagar, Gujarat**

**GOVERNMENT ENGINEERING COLLEGE,**

**GANDHINAGAR**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION**



***CERTIFICATE***

# Certificate

This is to certify that Mr./Ms. **HARDIK** **BHAMMAR**

Enrollment No **210130111013** of B.E. Semester 5 Electronics & Communication Engineering of this Institute (GTU Code: **13** ) has satisfactorily completed the Practical / Term work for the subject **Python Programming (3151108)** for the academic year 2023\_-24\_.

Place: Date:

### Name and Sign of Faculty member

**Head of the Department**

Python Programming (3151108)

**Vision of E&C Department:**

* To produce globally competitive and resourceful EC Engineers to bring out research, innovation and teamwork for development in various fields of Electronics and Communication Engineering.

**Mission of E&C Department:**

* To impart quality education in the fields of Electronics and Communication Engineering through effective teaching-learning processes, well equipped laboratory facilities and collaborative ventures with industries.
* To provide the students of E&C engineering an environment of an academic freedom that will ensure the exchange of ideas and the dissemination of knowledge in this discipline.
* To provide the platform for students to perform the project work related to needs of society.

**Program Educational Objectives (PEOs) of E&C Department:**

1. To prepare graduates to excel in postgraduate programs or pursue higher studies or to succeed in industry / technical profession through global, rigorous education.
2. To train graduates with good scientific and engineering breadth so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
3. To inculcate in graduates professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, and an ability to relate engineering issues to broader social context.

**Program Specific Outcomes (PSOs) of E&C Department:**

**PSO-I:** The ability to implement and apply knowledge of Electronics and Communication Engineering subjects in the analysis, design and development of various types of electronic and  communication systems.

**PSO-II:** Demonstrate soft skills, aptitude and technical skills & knowledge to work in the industry and Research and Development organization.

**Program Outcomes (POs):**

NBA has defined the following twelve POs for an engineering graduate:

|  |
| --- |
| 1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| 1. **Problem Analysis**: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences. |
| 1. **Design/Development of Solutions**: 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. |
| 1. **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 for complex problems:   Φthat cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline as against problems given at the end of chapters in a typical text book that can be solved using simple engineering theories and techniques.  Φthat may not have a unique solution. For example, a design problem can be solved in many ways and lead to multiple possible solutions;  Φthat require consideration of appropriate constraints / requirements not explicitly given in the problem statement such as cost, power requirement, durability, product life, etc.;  Φwhich need to be defined (modelled) within appropriate mathematical framework; and  Φthat often require use of modern computational concepts and tools, for example, in the design of an antenna or a DSP filter. |
| 1. **Modern Tool Usage**: 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. |
| 1. **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. |
| 1. **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. |
| 1. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| 1. **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| 1. **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. |
| 1. **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. |
| 1. **Life-long Learning**: Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change. |

**Course Outcomes (COs):**

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

### Index

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Objective(s) of Experiment** | **Date** | **Sign.** | **Remarks** |
| 1. | Write Python programs to understand control  structures |  |  |  |
| 2. | Write Python programs to understand list and  tuples |  |  |  |
| 3. | Use conditional statements and loops in Python  programs |  |  |  |
| 4. | Write python programs to create functions and use  functions in the program |  |  |  |
| 5. | Import module and use it in Python programs |  |  |  |
| 6. | Write python program to plot data using PyPlot |  |  |  |
| 7. | To become familiar with MicroPython and NodeMCU. Configure NodeMCU for  MicroPython. |  |  |  |
| 8. | Write program in MicroPython to send digital data on GPIO pins of NodeMCU and glow LED connected with NodeMCU or any other  MicroPython supported board. |  |  |  |
| 9. | Connect Digital/Analog I/O module with NodeMCU and write program to display  temperature in MicroPython. |  |  |  |
| 10. | 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 or any other cloud  service) |  |  |  |

### Experiment No: 1

**Write Python programs to understand control structures**

**Competency and Practical Skills:** Pyhton Programming

Relevant CO: CO1

**Objectives:** To get familiarity with basic step to run python program.

**Equipment/Instruments:** Computer and Python Compiler and IDE

Theory:

Control structures are an essential part of programming that allows you to specify the flow of execution in your code. In Python, there are several types of control structures including:

1. **Conditional statements**: These statements allow you to execute certain code blocks only if a certain condition is met. For example, you might want to check if a number is even or odd before performing some operation on it. In Python, the conditional statements are if, elif, and else.
2. **Loops:** Loops allow you to execute a block of code multiple times. Python has two types of loops: for loops and while loops. For loops are used to iterate over a sequence of elements (such as a list), while while loops are used to repeat a block of code as long as a certain condition is true.
3. **Exception handling:** Exception handling allows you to handle errors that may occur in your code gracefully, without causing the program to crash. In Python, you can use the try and except statements to handle exceptions.

Control structures are an important concept in programming, as they allow you to write code that can adapt to different input, makes decisions based on conditions, and repeat actions. By using control structures effectively, you can write more powerful and flexible programs.

Procedure:

1. Enter program codes as demonstrated in the sample code below, prepare a file/script and run the program(s).(One may prepare flow chart and pseudo code, if required before entering the code.)

Sample codes for if else Statement & Results:

x = 5 if x > 5:

print("x is greater than 5") elif x == 5:

print("x is equal to 5")

In this example, the condition x > 5 is false, so the code block print(“x is greater than 5”) will not be executed. The condition x > 5 is false, so the condition x == 5 would be checked, and the code block print(“x is equal to 5”) is executed because x = 5.

Sample code of for loops

A for loop in Python iterates over a sequence of elements, such as a list or a string. Here is the general syntax for a for loop:

for element in sequence: # code to be executed

Here is an example of a for loop that iterates over a list of numbers and prints out each number:

numbers = [1, 2, 3, 4, 5] for number in numbers:

print(number) Output:

1

2

3

4

5

You can also use the range() function to specify the number of iterations in the loop. The range()

function returns a sequence of numbers, starting from 0 by default, and increments by 1 (also by default), and ends at a specified number.Here is an example of using range() in a for loop:

for i in range(5): print(i)

Output: 0

1

2

3

4

You can also specify a start and end value for the range() function, like this:

for i in range(2, 5): print(i)

Output: 2

3

4

You can even specify a step value, which determines how much the loop variable is incremented

each iteration. For example, the following for loop will count by twos:

for i in range(0, 10, 2): print(i)

Output: 0

2

4

6

8

Sample code of while loops

A while loop in Python executes a block of code repeatedly until a certain condition is met. The syntax for a while loop is as follows:

while condition:

# code to be executed

Here is an example of a while loop that counts from 1 to 5:

i = 1

while i <= 5:

print(i) i += 1 Output:

1

2

3

4

5

It’s important to include a way to change the value of the loop condition, or else the loop will run

forever (called an “infinite loop”). In the example above, we increase the value of i by 1 each iteration using the += operator.

**Break and continue**

In Python, the break statement is used to exit a loop prematurely. When the breakstatement is encountered in the code, the program control immediately exits the current loop, and continues with the next statement after the loop. This allows you to exit a loop early, when a certain condition is met, rather than continuing to iterate until the end of the loop.

Here is an example of using the break statement in a for loop:

for i in range(10): if i == 5:

break print(i)

Output: 0

1

2

3

4

In this example, the loop iterates over the range from 0 to 9, but the break statement exits the loop

when i is equal to 5, so only the numbers from 0 to 4 are printed. It’s also possible to use break in while loop, such as:

i = 0

while i < 10: i += 1

if i == 5:

break print(i)

Output: 1

2

3

4

As you can see here the while loop only runs 4 times and breaks out of the loop when i equal to 5.

It’s important to be careful when using break statement as if it's not used properly it might lead to an infinite loop or miss certain conditions, making the code to stop running even when it should not. Also, note that it only breaks out of the nearest enclosing loop and the control goes to the next statement after the loop.

In Python, the continue statement is used to skip the current iteration of a loop, and continue with the next iteration. When the continue statement is encountered in the code, the current iteration of the loop is immediately terminated, and the program control proceeds to the next iteration of the loop. This allows you to selectively skip certain iterations of the loop, rather than completely ending the loop altogether.

Here is an example of using the continue statement in a for loop:

for i in range(10): if i % 2 == 0:

continue print(i)

Output: 1

3

5

7

9

In this example, the loop iterates over the range from 0 to 9, but the continue statement skips all

the even numbers (i.e., those for which i % 2 == 0 is true), and only odd numbers are printed. It’s also possible to use continue in while loop.

i = 0

while i < 10: i += 1

if i % 2 == 0: continue

print(i) Output:

1

3

5

7

9

As you can see the continue statement here skips the even number which makes the loop only

print the odd numbers starting from 1

It’s important to be careful when using **continue** statement as if it's not used properly it might lead to an infinite loop and unexpected results.

**Sample code of Exceptions handling in Python**

Exceptions in Python are a mechanism to handle errors and exceptional conditions in the program execution. An exception is raised (or thrown) when a certain condition or error occurs in the code. When an exception is raised, the program execution stops and the program control jumps to the nearest exception handler. If an exception is not handled, it will cause the program to terminate.

To handle exceptions in Python, you use a try-except block. The code that may raise an exception is placed in the try block, and the code to handle the exception is placed in the except block. Here's an example:

try:

x = 1 / 0

except ZeroDivisionError: print("division by zero")

In this example, a ZeroDivisionError exception is raised when trying to divide 1 by 0, which is caught by the except block and a message is printed.

You can also have multiple except block to handle multiple exception.

try:

x = 1 / 0

except ZeroDivisionError: print("division by zero")

except Exception as e:

print("an exception occurred: ", e)

It’s important to note that the order of the except block matter, Python will execute the first except block that it finds that matches the type of exception that was raised.

Conclusion:

### Experiment No: 2

**Write Python programs to understand list and tuples**

**Competency and Practical Skills:** Pyhton Programming

Relevant CO: CO1

**Objectives:** To get familiarity list and tuple in python program.

**Equipment/Instruments:** Computer and Python Compiler and IDE

Theory:

The important characteristics of Python lists are as follows:

* Lists are ordered.
* Lists can contain any arbitrary objects.
* List elements can be accessed by index.
* Lists can be nested to arbitrary depth.
* Lists are mutable.
* Lists are dynamic.

Creating a list

A list is a collection of items that are ordered and changeable. You can create a list in Python by enclosing a comma-separated sequence of items in square brackets. For example:

fruits = ["apple", "banana", "cherry"]

You can also create an empty list by calling the list() constructor without any arguments.

numbers = list()

**Accesing a list**

You can access items in a list by referring to their index number. For example:

print(fruits[1]) # prints "banana"

You can also use negative indexing to access items from the end of the list. For example:

print(fruits[-1]) # prints "cherry"

**Modifying a List**

You can add items to a list using the append() method. For example:

fruits.append("orange")

print(fruits) # prints ['apple', 'banana', 'cherry', 'orange']

You can insert an item at a specific index using the insert() method. For example:

fruits.insert(1, "mango")

print(fruits) # prints ['apple', 'mango', 'banana', 'cherry', 'orange']

You can remove items from a list using the remove() method. For example:

fruits.remove("banana")

print(fruits) # prints ['apple', 'cherry', 'orange'] fruits.pop(1)

print(fruits) # prints ['apple', 'cherry', 'orange']

You can also use the del statement to remove items by index. For example:

del fruits[1]

print(fruits) # prints ['apple', 'orange']

List slicing

You can access a range of items in a list by slicing it. Slicing is done by specifying the start and end indices separated by a colon. For example:

fruits = ["apple", "banana", "cherry"] print(fruits[1:3]) # prints ["banana", "cherry"]

You can also leave out the start or end index to slice from the beginning or to the end of the list respectively. For example:

print(fruits[:2]) # prints ["apple", "banana"] print(fruits[1:]) # prints ["banana", "cherry"]

List Methods

There are various built-in methods available for working with lists, such as:

* len(): Returns the number of items in the list.
* count(): Returns the number of occurrences of a specific item in the list.
* index(): Returns the index of the first occurrence of a specific item in the list.
* sort(): Sorts the items in the list in ascending order.
* reverse(): Reverses the order of the items in the list.
* extend(): Adds multiple items to the list.

Example:

fruits = ["apple", "banana", "cherry"] print(len(fruits)) # prints 3 print(fruits.count("banana")) # prints 1 print(fruits.index("cherry")) # prints 1 fruits.sort()

print(fruits) # prints ["apple", "banana", "cherry"] fruits.reverse()

print(fruits) # prints ["cherry", "banana", "apple"] fruits.extend(["orange", "mango"])

print(fruits) # prints ["cherry", "banana", "apple", "orange", "mango"]

**Tuples**

Tuples are similar to lists in Python, but they are immutable, meaning that their items cannot be changed once they are created. This manual will cover the basics of working with tuples in Python.

Creating a Tuple

You can create a tuple in Python by enclosing a comma-separated sequence of items in parentheses. For example:

coordinates = (4, 5)

You can also create an empty tuple by calling the tuple() constructor without any

arguments.print(coordinates[0]) # prints 4

empty\_tuple = tuple()

Accessing Items

You can access items in a tuple by referring to their index number, just as you would with a list. For example:

print(coordinates[0]) # prints 4

Modifying a Tuple

As mentioned earlier, tuples are immutable, so you cannot add or remove items from a tuple. However, you can assign a new value to a variable that holds a tuple, but the tuple itself will remain unchanged.

Tuple Methods

There are a few built-in methods available for working with tuples, such as: len(): Returns the number of items in the tuple.

count(): Returns the number of occurrences of a specific item in the tuple. index(): Returns the index of the first occurrence of a specific item in the tuple. Examples:

coordinates = (4, 5) print(len(coordinates)) # prints 2 print(coordinates.count(5)) # prints 1 print(coordinates.index(5)) # prints 1

Tuple Packing

You can also unpack the items of a tuple into separate variables. For example:

x, y = coordinates print(x) # prints 4 print(y) # prints 5

You can also create a tuple by packing multiple items together.

coordinates = 4, 5

Converting Tuples

You can convert a tuple to a list using the list() constructor.

coordinates\_list = list(coordinates)

You can convert a list to a tuple using the tuple() constructor.

fruits\_tuple = tuple(fruits)

It’s important to note that the tuple is generally used when you want to ensure that a certain

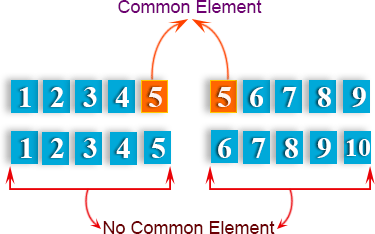
collection of items will remain unchanged, while a list is typically used when you need to make changes to the items in a collection.

Procedure:

1. Enter program codes as demonstrated in the sample code below, prepare a file/script and run the program(s).(One may prepare flow chart and pseudo code, if required before entering the code.)

Sample codes & Results:

**Write a Python function that takes two lists and returns True if they have at least one common member.**



**Python Code:**

def common\_data(list1, list2): result = False

for x in list1:

for y in list2: if x == y:

result = True return result

print(common\_data([1,2,3,4,5], [5,6,7,8,9]))

print(common\_data([1,2,3,4,5], [6,7,8,9]))

**Output:**

True

False

Write a Python program to check whether an element exists within a tuple.

**Python Code:**

tuplex = ("w", 3, "r", "e", "s", "o", "u", "r", "c", "e") print("r" in tuplex)

print(5 in tuplex)

**Sample Output:**

True

False

Conclusion:

### Experiment No: 3

**Use conditional statements and loops in Python programs**

**Competency and Practical Skills:** Pyhton Programming

Relevant CO: CO1

**Objectives:** To get familiarity with basic step to run python program.

**Equipment/Instruments:** Computer and Python Compiler and IDE

**Theory:** Theory related to this experiment is already mention in the experiment no.1

Procedure:

1. Enter program codes as demonstrated in the sample code below, prepare a file/script and run the program(s).(One may prepare flow chart and pseudo code, if required before entering the code.)

Sample codes & Results:

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

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

**Python Code:**

n=5;

for i in range(n):

for j in range(i): print ('\* ', end="")

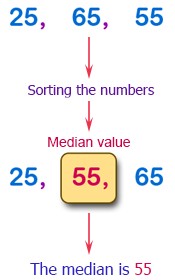
print('')

for i in range(n,0,-1): for j in range(i):

print('\* ', end="") print('')

Write a Python program to find the median of three values.

#### Pictorial Presentation:



**Python Code:**

a = float(input("Input first number: ")) b = float(input("Input second number: ")) c = float(input("Input third number: ")) if a > b:

if a < c:

median = a elif b > c:

median = b else:

median = c

else:

if a > c:

median = a elif b < c:

median = b else:

median = c

print("The median is", median)

#### Output:

input first number: 30

Input second number: 50

Input third number: 20

The median is 30.0

Conclusion:

### Experiment No: 4

**Write python programs to create functions and use functions in the**

**Competency and Practical Skills:** Pyhton Programming

Relevant CO: CO1

**Objectives:** To get familiarity with creating function in python program. **Equipment/Instruments:** Computer and Python Compiler and IDE **Theory:**

What is a function? What does a function do? As the name suggests function does a specific task. This task can be used repeatedly or once as per the use case. The main purpose of a function is to separate the logic into different blocks. We can create our functions or use any of the available in- built functions in python.

Syntax:

def function\_name(parameters): """docstring""" statement(s)

return expression

We can see in the syntax that to create a function we need to use the def keyword following the

function name. Parameter(s)/ Argument(s) is the only way through which we can pass data to the function. We can add a **docstring** to explain the use of function and its parameter(s)**.** A function can return a value if desired. It is not compulsory to return something.

Once we define the function, to use it we need to call that function. Let’s assume I have created a function named odd\_or\_even. To call this function, I can write like odd\_or\_even(). And if this function is expecting some argument(s), I can write it like odd\_or\_even(2).

Sum of Two Numbers

If we want to just print the sum of two numbers or just to see the sum of two numbers, we do not need to return anything. And if we try to assign it to a variable we get None as the value.

def sum\_num(num1, num2): total = num1 + num2

print(total)sum\_num(5,10)# OUTPUT: 15def sum\_num(num1, num2):

total = num1 + num2 print(total)temp = sum\_num(5,10)

print(temp)""" OUTPUT: 15

None """

Now, if we want to return the sum then we can do using the following code.

def sum\_num(num1, num2): total = num1 + num2

return totaltemp = sum\_num(5,10) print(temp)# OUTPUT: 15

Keyword arguments in Python

In general, we need to pass arguments in a specific order to a function. But, using keyword arguments we can pass arguments in any order. Let’s understand it by an example of division. In the example below, if we want 3/2 then we will write divide(3,2).

def divide(num1, num2):

return num1/num2ans = divide(3,2) print(ans)# OUTPUT: 1.5

But now, by mistake, we wrote divide(2,3). This error can make the whole logic wrong.

def divide(num1, num2):

return num1/num2ans = divide(2,3) print(ans)# OUTPUT: 0.6666666666666666

To save yourself from this kind of mistake we can use keyword arguments. This is just a normal argument passing, but instead of passing only value, we will pass the variable name as well.

def divide(num1, num2):

return num1/num2ans = divide(num1=3, num2=2) print(ans)# OUTPUT: 1.5

So, now we can pass data in any order and this way logic will be intact.

def divide(num1, num2):

return num1/num2ans = divide(num2=2, num1=3) print(ans)# OUTPUT: 1.5

Procedure:

**1.** Enter program codes as demonstrated in the sample code below, prepare a file/script and run the program(s).(One may prepare flow chart and pseudo code, if required before entering the code.)

Sample codes & Results:

1. Write a Python function to find the maximum of three numbers.

**Python Code:**

def max\_of\_two( x, y ): if x > y:

return x return y

def max\_of\_three( x, y, z ):

return max\_of\_two( x, max\_of\_two( y, z ) ) print(max\_of\_three(3, 6, -5))

**Output:**

6

Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

#### Python Code:

def factorial(n): if n == 0:

return 1 else:

return n \* factorial(n-1)

n=int(input("Input a number to compute the factiorial : ")) print(factorial(n))

**Output:**

Input a number to compute the factiorial : 7

5040

Conclusion:

### Experiment No: 5

**Write a program to import module and use it in Python programs**

**Competency and Practical Skills:** Python Programming

Relevant CO: CO1

**Objectives:** To learn about usage of module in python.

**Equipment/Instruments:** Computer and Python Compiler and IDE

Theory:

In Python, a module is a single file containing Python definitions and statements. A package is a collection of modules that are organized in a directory hierarchy. Modules and packages allow for a way to organize and reuse code, making it easier to build and maintain large projects. They also provide a way to share code with others and to use code written by others. In Python, modules and packages can be imported and used in other scripts, allowing for efficient code reuse and modular design.

**Importing modules**

1. Importing a module in Python is done using the import statement. To import a module, simply write import followed by the name of the module you want to import. For example, to import the math module, you would write import math.
2. Once a module is imported, you can access its functions and variables using the dot notation. For example, to use the sqrt() function from the math module, you would write math.sqrt(4) which would return 2.0.
3. You can also import specific functions or variables from a module using the from keyword. For example, to import only the sqrt() function from the math module, you would write from math import sqrt. Then you can use the sqrt function directly like sqrt(4) which would return 2.0
4. If you want to import multiple functions or variables from a module, you can list them separated by commas. For example, from math import sqrt, pi.
5. You can also use the as keyword to give a module or function a different name when importing it. This is useful if the name of the module or function you are importing conflicts with the name of an existing variable or function in your code. For example, import math as m and then you can use the functions with m.sqrt(4)
6. You can also use \* to import all the functions or variables from a module. For example from math import \*. But it is not recommended as it may cause naming conflicts and make it harder to understand where a particular function is coming from.
7. Once you have imported a module, you can check which functions and variables it contains using the dir() function. For example, dir(math) will return a list of all the functions and variables in the math module.

Here are some examples of importing modules in Python:

* Importing the math module:

import math

print(math.sqrt(4)) # prints 2.0

* Importing the random module and using the random() function to generate a random float between 0 and 1:

import random print(random.random())

* Importing the datetime module and using the datetime.now() function to get the current date and time:

import datetime

now = datetime.datetime.now() print(now)

* Importing specific functions or variables from a module:

from math import sqrt, pi print(sqrt(4)) # prints 2.0 print(pi) # prints 3.141592653589793

* Importing all functions or variables from a module:

from math import \* print(sqrt(4)) # prints 2.0

print(pi) # prints 3.141592653589793

* Importing a module and giving it a different name:

import math as m print(m.sqrt(4)) # prints 2.0

* Using the dir() function to see what's inside a module:

import math print(dir(math))

It’s worth noting that it’s best practice to import only what you need and avoid using from module

import \* as it can cause naming conflicts and make it harder to understand where a particular function is coming from.

**Standard library modules**

Python’s Standard Library is a collection of modules that are included with the Python programming language. These modules provide a wide range of functionality, from basic data types and file I/O to more advanced features such as networking and web development. We saw some examples in the previous section, other libraries are:

1. math **module**: This module provides mathematical functions and constants, including trigonometric and logarithmic functions, as well as the mathematical constant pi.
2. os **and** os.path **modules**: The os module provides a way to interact with the operating system, including functions for working with files and directories. The os.path module provides additional functionality for working with file paths.
3. sys **module**: The sys module provides access to various system-specific parameters and functions, including the interpreter's version and exit status.
4. json **module**: The json module provides functions for working with JSON data, including encoding and decoding.
5. datetime **module**: The datetime module provides classes for working with dates and times, including date, time, and datetime classes.
6. random **module**: The random module provides functions for generating random numbers, including the ability to generate random integers, floats, and selections from a list.
7. re **module**: The re module provides functions for working with regular expressions, including the ability to search and replace text using regular expressions.
8. urllib **module**: The urllib module provides a set of simple, high-level APIs for working with URLs and HTTP requests.
9. zipfile **and** tarfile **modules**: These modules provide functions for working with ZIP and TAR archives, respectively.
10. collections **module**: The collections module provides alternatives to built-in types that can be more efficient in certain cases. It includes alternatives such as OrderedDict, defaultdict, and Counter.

This is a general manual of some of the standard modules of python, you can find more information in the official documentation and explore the functionality of each module in more detail.

There are also libraries for specific fields, for example, these libraries for data science and machine learning:

1. numpy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
2. pandas: Pandas is a library that provides easy-to-use data structures and data analysis tools for the Python programming language. It is widely used for data manipulation and cleaning.
3. scipy: SciPy is a library for the Python programming language that supplements the popular NumPy library with additional algorithms such as optimization, interpolation, signal and image processing, and more.
4. Scikit-learn: Scikit-learn is a library for the Python programming language that provides simple and efficient tools for data mining and data analysis. It is built on top of NumPy and SciPy and includes popular algorithms for supervised and unsupervised learning.
5. tensorflow: TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.
6. keras: Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. Keras provides a user-friendly API for building and training deep learning models.
7. matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Installing libraries in Python using pip is a straightforward process. Pip is a package management system for Python that allows you to easily install, upgrade, and remove packages from your system.

Here are the steps for installing a library in Python using pip:

1. Open a command prompt or terminal window on your system.
2. Make sure pip is up to date by running the command pip install --upgrade pip
3. To install the matplotlib library, use the command pip install matplotlib.
4. If you want to install a specific version of the matplotlib library, you can use the command pip install matplotlib==version\_number. For example, pip install matplotlib==3.3.2
5. If you want to install the matplotlib library in a specific environment, you can use the command pip install matplotlib -t /path/to/environment
6. After the installation has finished, you can import the matplotlib library in your script and start using it.

Note that matplotlib may have additional dependencies that need to be installed, such as numpy, so pip will automatically install them as well. If you are installing the matplotlib library in a shared environment, you may need to use sudo pip install matplotlib to install the library with administrator privileges.

It’s also worth mentioning that you can use pip to manage your library dependencies, you can create a requirements.txt file with the libraries and versions you need and use the command pip install -r requirements.txt to install all the libraries in one command.

Procedure:

1. Enter program codes as demonstrated in the sample code below, prepare a file/script and run the program(s).(One may prepare flow chart and pseudo code, if required before entering the code.)

Sample codes & Results:

Write a Python program that generates random alphabetical characters, alphabetical strings, and alphabetical strings of a fixed length.

Use random.choice

**Python Code:**

import random import string

print("Generate a random alphabetical character:") print(random.choice(string.ascii\_letters)) print("\nGenerate a random alphabetical string:") max\_length = 255

str1 = ""

str1

i

for

in range(random.randint(1, max\_length)):

+= random.choice(string.ascii\_letters)

print(str1)

print("\nGenerate a random alphabetical string of a fixed length:")

str1 = ""

for i in range(10):

str1 += random.choice(string.ascii\_letters) print(str1)

Write a Python program to shuffle the elements of a given list. Use random.shuffle()

#### Python Code:

import random

nums = [1, 2, 3, 4, 5]

print("Original list:") print(nums) random.shuffle(nums) print("Shuffle list:") print(nums)

words = ['red', 'black', 'green', 'blue'] print("\nOriginal list:")

print(words) random.shuffle(words) print("Shuffle list:") print(words)

**Output:**

Original list:

[1, 2, 3, 4, 5]

Shuffle list:

[5, 1, 4, 2, 3]

Original list:

['red', 'black', 'green', 'blue']

Shuffle list:

['blue', 'green', 'red', 'black']

Conclusion:

Quiz (tick right answer)

* 1. Which of these definitions correctly describes a module?
     1. Denoted by triple quotes for providing the specification of certain program elements
     2. Design and implementation of specific functionality to be incorporated into a program
     3. Defines the specification of how it is to be used
     4. Any program that reuses code
  2. Which of the following is not an advantage of using modules?
     1. Provides a means of reuse of program code
     2. Provides a means of dividing up tasks
     3. Provides a means of reducing the size of the program
     4. Provides a means of testing individual parts of the program
  3. Program code making use of a given module is called a of the module.
     1. Client
     2. Docstring
     3. Interface
     4. Modularity
  4. All modular designs are because of a top-down design process.
     1. True
     2. False

### Experiment No: 06

**Write python program to plot data using PyPlot**

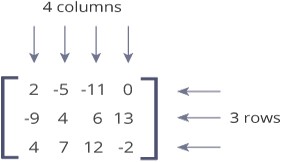
**Competency and Practical Skills:** Pyhton Programming

**Relevant CO:** CO2,CO3

**Objectives:** To get familiarity PyPlot and learn to plot various graphs. **Equipment/Instruments:** Computer and Python Compiler and IDE **Theory:-**

**Python Matrices:**

A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.

For example: 

This matrix is a 3x4 (pronounced "three by four") matrix because it has 3 rows and 4 columns.

Python doesn't have a built-in type for matrices. However, we can treat list of a list as a matrix. For example:

A = [[1, 4, 5],

[-5, 8, 9]]

We can treat this list of a list as a matrix having 2 rows and 3 columns.

**Python Matrix Example**

*A = [[1, 4, 5, 12],*

*[-5, 8, 9, 0],*

*[-6, 7, 11, 19]]*

*print("A =", A) print("A[1] =", A[1])*

*# 2nd row*

*print("A[1][2] =", A[1][2]) # 3rd element of 2nd row print("A[0][-1] =", A[0][-1]) # Last element of 1st Row column = []; # empty list*

*for row in A: column.append(row[2])*

*print("3rd column =", column)*

**Output:**

A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]

A[1] = [-5, 8, 9, 0]

A[1][2] = 9

A[0][-1] = 12

3rd column = [5, 9, 11]

**Plotting in Python:**

Python has the ability to create graphs by using the matplotlib library. It has numerous packages and functions which generate a wide variety of graphs and plots. It is also very simple to use. It along with

numpy and other python built-in functions achieves the goal.

Here we take a mathematical function to generate the x and Y coordinates of the graph. Then we use matplotlib to plot the graph for that function. Here we can apply labels and show the title of the graph as shown below. We are plotting the graph for the trigonometric function − tan.

Example

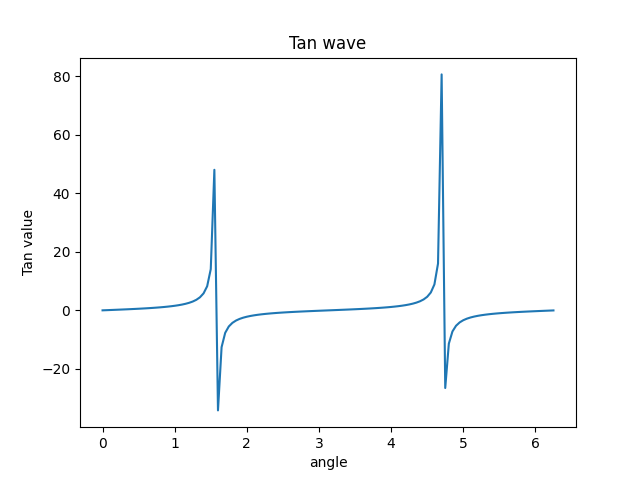
from matplotlib import pyplot as plt import numpy as np

import math #needed for definition of pi x = np.arange(0, math.pi\*2, 0.05)

y = np.tan(x) plt.plot(x,y) plt.xlabel("angle") plt.ylabel("Tan value") plt.title('Tan wave')

plt.show

**Output:**



Multiplots:

We can have two or more plots on a single canvas by creating multiple axes and using them in the program.

Example

import matplotlib.pyplot as plt import numpy as np

import math

x = np.arange(0, math.pi\*2, 0.05) fig=plt.figure()

axes1 = fig.add\_axes([0.1, 0.1, 0.8, 0.8]) # main axes

axes2 = fig.add\_axes([0.55, 0.55, 0.3, 0.3]) # inset axes

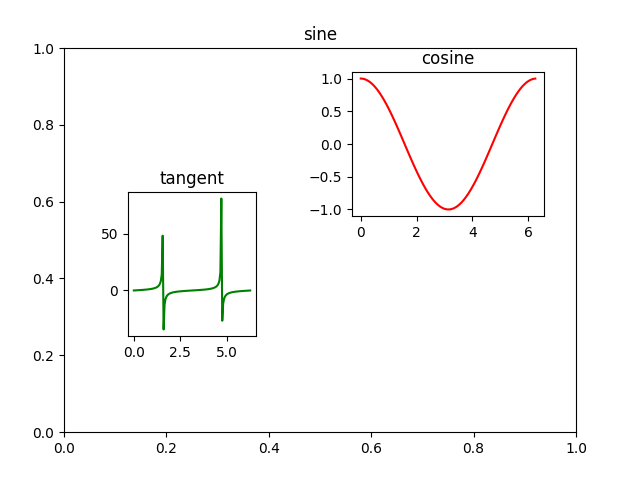
axes3 = fig.add\_axes([0.2, 0.3, 0.2, 0.3]) # inset axes axes1.plot(x, np.sin(x), 'b')

axes2.plot(x,np.cos(x),'r')

axes3.plot(x,np.tan(x),'g') axes1.set\_title('sine') axes2.set\_title("cosine") axes3.set\_title("tangent")

plt.show()

**Output:**



Conclusion:

### Experiment No: 07

### 

**To become familiar with MicroPython and NodeMCU. Configure NodeMCU for MicroPython.**

**Competency and Practical Skills:** Pyhton Programming

**Relevant CO:** CO1, CO2, CO3,CO4

**Objectives:** To become familiar with MicroPython and NodeMCU. Configure NodeMCU for MicroPython.

**Equipment/Instruments:** NodeMCU 12E ESP8266 Board ,USB to Micro USB cable, Software- PuTTY, PyFlasher

**Theory:-**

Micropython:

MicroPython is an implementation of Python 3, targeted for embedded systems and microcontrollers. MicroPython and regular Python are very similar, so if you already know how to write programs in Python, you already know how to program in MicroPython.

Micropython includes a small subset of the Python standard library, so not every module from the standard library is available. However, MicroPython does include modules to access low-level hardware, meaning there are modules to easily interact with GPIOs on microcontrollers.

MicroPython also comes with an interactive REPL (Read-Evaluate-Print Loop), which is an often overlooked amazing feature of MicroPython. The REPL allows you to connect to a microcontroller, execute code quickly without the need to compile or upload code. Which gives immediate feedback on whether your program is working as intended.

ESP32/ESP8266:

The ESP32 and ESP8266 are low-cost Wi-Fi modules, perfect for DIY Internet of Things (IoT) projects.Both come with general purpose input/output pins (GPIOs) and support a variety of protocols, including SPI, I2C, UART, and more. The most attractive part of the ESP range is that they come with wireless networking, separating them from their Arduino microcontroller counterparts. All in all, the ESP series allows you to easily control/monitor devices remotely using Wi-Fi for a very low price.

For this Experiment, we’ll be using the NodeMCU 12-E. We’ll be referencing this board and its

instructions for this particular Experiment.



The NodeMCU 12E ESP8266 Board.

### Procedure:

1. **Download the MicroPython Firmware**

Before we connect the NodeMCU board to our PC, we need to download the most recent MicroPython firmware and load it onto the NodeMCU. You can download it from the MicroPython download page (https://micropython.org/download/#esp8266).

On the downloads page you’ll have 3 main choices:

* + Stable firmware builds for 1024kb modules and above,
  + Daily firmware builds for 1024kb modules and above,
  + Daily firmware builds for 512kb modules. Select firmware builds for this Experiment.

1. **Load Firmware Onto the NodeMCU**

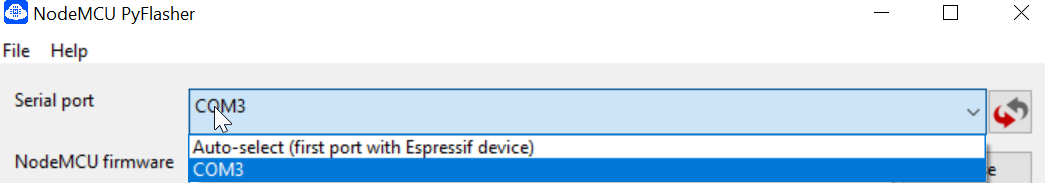
Once you have the firmware downloaded, it’s time to load it onto the NodeMCU board. There are two main steps to achieve this:

* + Put NodeMCU into boot-loader mode
  + Copy across firmware to NodeMCU

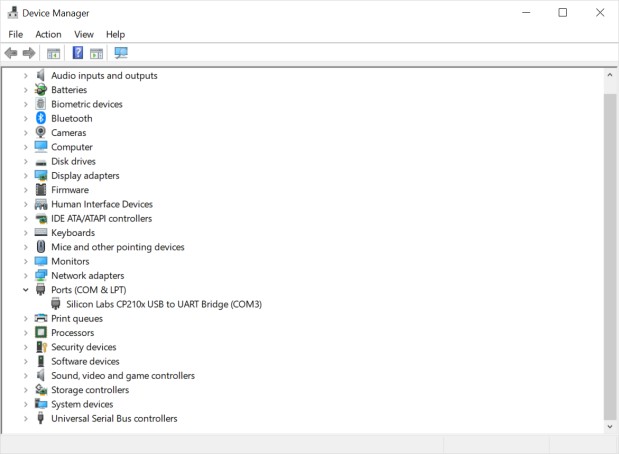
To flash the MicroPython firmware onto the NodeMCU, you can choose following technique.

1. **NodeMCU PyFlasher**

A self-contained NodeMCU flasher with GUI that utilizes esptool.py and wxPython (Python GUI library), developed by Marcel Stör. Use the latest release, which is available for Windows (.exe), macOS (.dmg) and anything that runs Python.

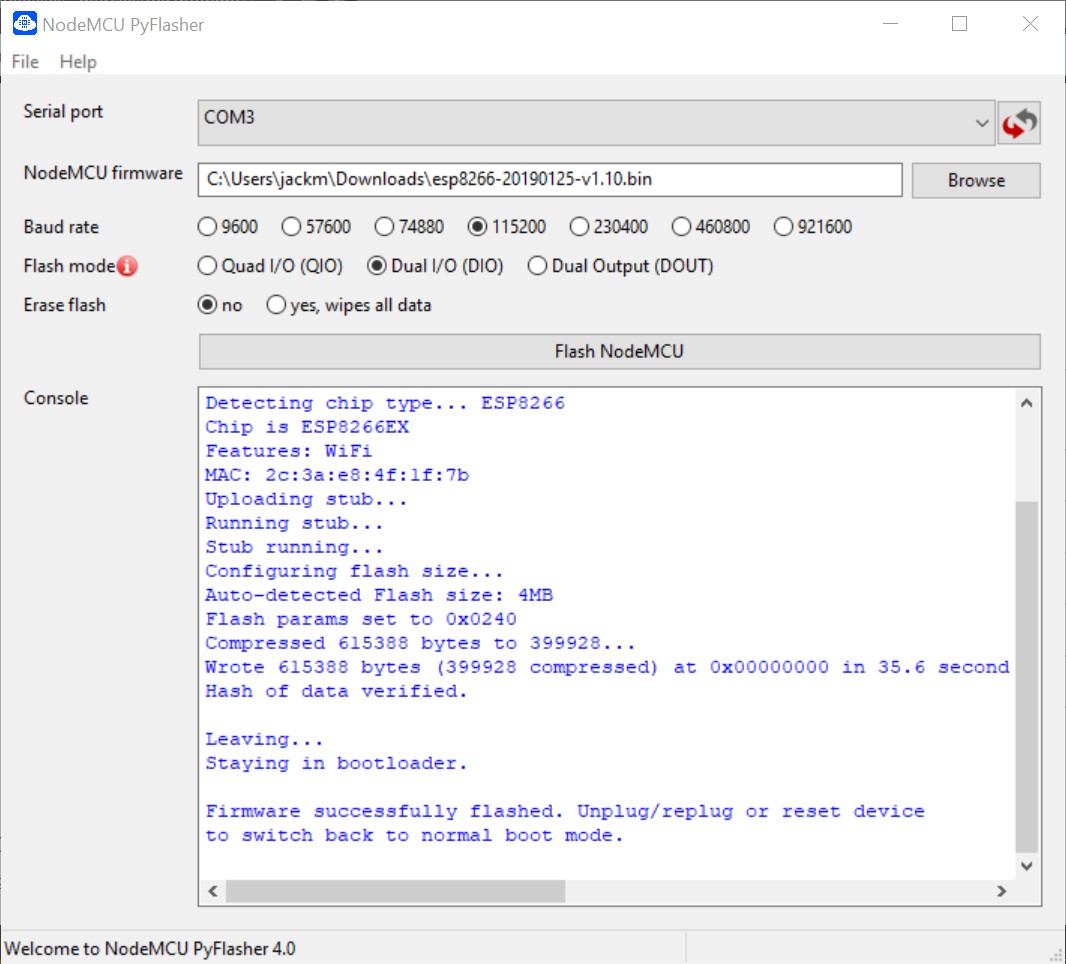
Once you’ve plugged your NodeMCU, you should now be able to select the COM port that it is connected on. If you don’t see it on this list, you may need to download the USB-Serial driver. Once the driver is active, you will able to determine the COM port number by either selecting it in the drop-down on NodeMCU PyFlasher or in your device manager (on Windows).

Determine the COM port number in the drop-down on ModeMCU PyFlasher



Select the COM port in your device manager if you're using Windows.

Now just select your downloaded MicroPython firmware .bin file from the download location, select a baud rate of 115200 (default for NodeMCU), using a dual I/O flash mode and click Flash NodeMCU. After the process completes, you should see a screen like below.



The window you'll see once you've successfully completed the firmware download.

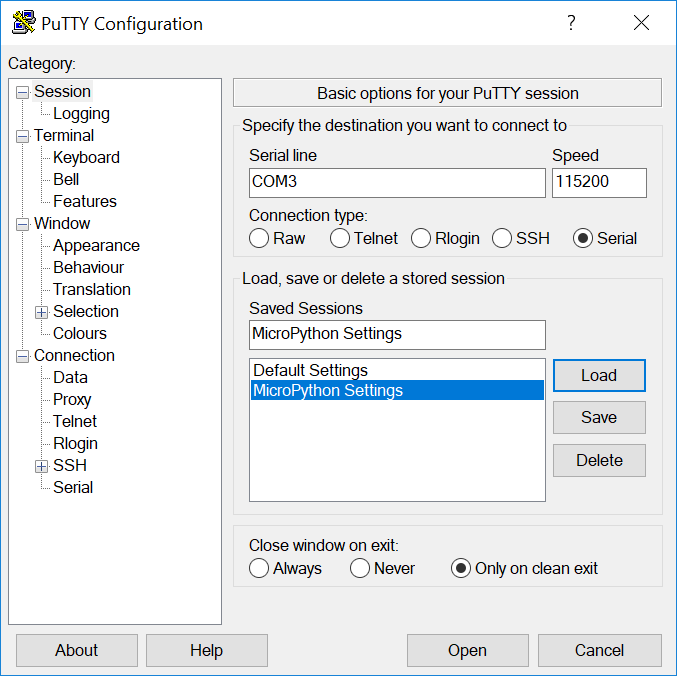
1. **Access the MicroPython REPL Prompt**

REPL (Read Evaluate Print Loop) is the name given to the interactive MicroPython prompt that you can access on the NodeMCU. Using the REPL greatly simplifies and speeds up testing out your code and running commands.

While you can access the REPL over Wi-Fi (WebREPL), this article will only go into how to access the REPL over a wired connection through the UART serial port.

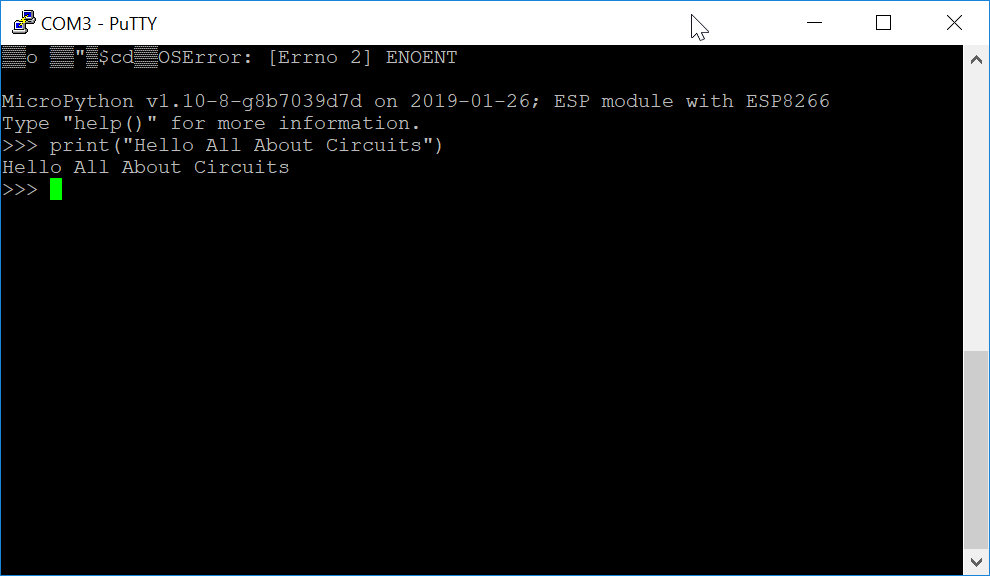
To access the prompt over USB-serial, you will need to use a terminal emulator program. PuTTY is an open source SSH and telnet client which you can use as a terminal emulator program to access your NodeMCU REPL, otherwise, there is TeraTerm on Windows, built-in screen program on Mac and Linux has picocom and minicom. There are many other terminal programs that will work, so choose your favorite.

Once you have opened PuTTY for the first time, you will need to set up the settings configuration to access the NodeMCU. First, we select a Serial connection type. Then, from step 3, we now know the COM port number that the NodeMCU board is on (COM3 for this example). From there, we can set a baud rate of 115200 and we are ready to connect. The settings can be saved as well for when you use it in the future. The settings for this example can be seen below.



The PuTTY configuration settings.

Once you connect you will be presented with a screen like below and can test out your code!



Once you're connected you'll be greeted with this screen.

Conclusion:

### Experiment No: 08

**Write program in MicroPython to send digital data on GPIO pins of NodeMCU and glow LED connected with NodeMCU or any other MicroPython supported board.**

**Competency and Practical Skills:** MicroPython Programming

**Relevant CO:** CO1, CO2,CO4

**Objectives:** To become familiar with MicroPython and NodeMCU. Configure NodeMCU for MicroPython.

**Equipment/Instruments:** NodeMCU 12E ESP8266 Board ,USB to Micro USB cable, Software- PuTTY, PyFlasher

Micropython:

MicroPython is an implementation of Python 3, targeted for embedded systems and microcontrollers. MicroPython and regular Python are very similar, so if you already know how to write programs in Python, you already know how to program in MicroPython.

Micropython includes a small subset of the Python standard library, so not every module from the standard library is available. However, MicroPython does include modules to access low-level hardware, meaning there are modules to easily interact with GPIOs on microcontrollers.

MicroPython also comes with an interactive REPL (Read-Evaluate-Print Loop), which is an often overlooked amazing feature of MicroPython. The REPL allows you to connect to a microcontroller, execute code quickly without the need to compile or upload code. Which gives immediate feedback on whether your program is working as intended.

ESP32/ESP8266:

The ESP32 and ESP8266 are low-cost Wi-Fi modules, perfect for DIY Internet of Things (IoT) projects.Both come with general purpose input/output pins (GPIOs) and support a variety of protocols, including SPI, I2C, UART, and more. The most attractive part of the ESP range is that they come with wireless networking, separating them from their Arduino microcontroller counterparts. All in all, the ESP series allows you to easily control/monitor devices remotely using Wi-Fi for a very low price.

For this Experiment, we’ll be using the NodeMCU 12-E. We’ll be referencing this board and its instructions for this particular Experiment.

The NodeMCU 12E ESP8266 Board.

### Procedure:

1. **Download the MicroPython Firmware**

Before we connect the NodeMCU board to our PC, we need to download the most recent MicroPython firmware and load it onto the NodeMCU. You can download it from the MicroPython download page (https://micropython.org/download/#esp8266).

1. **Load Firmware Onto the NodeMCU**

Once you have the firmware downloaded, it’s time to load it onto the NodeMCU board. There are two main steps to achieve this:

* + Put NodeMCU into boot-loader mode
  + Copy across firmware to NodeMCU

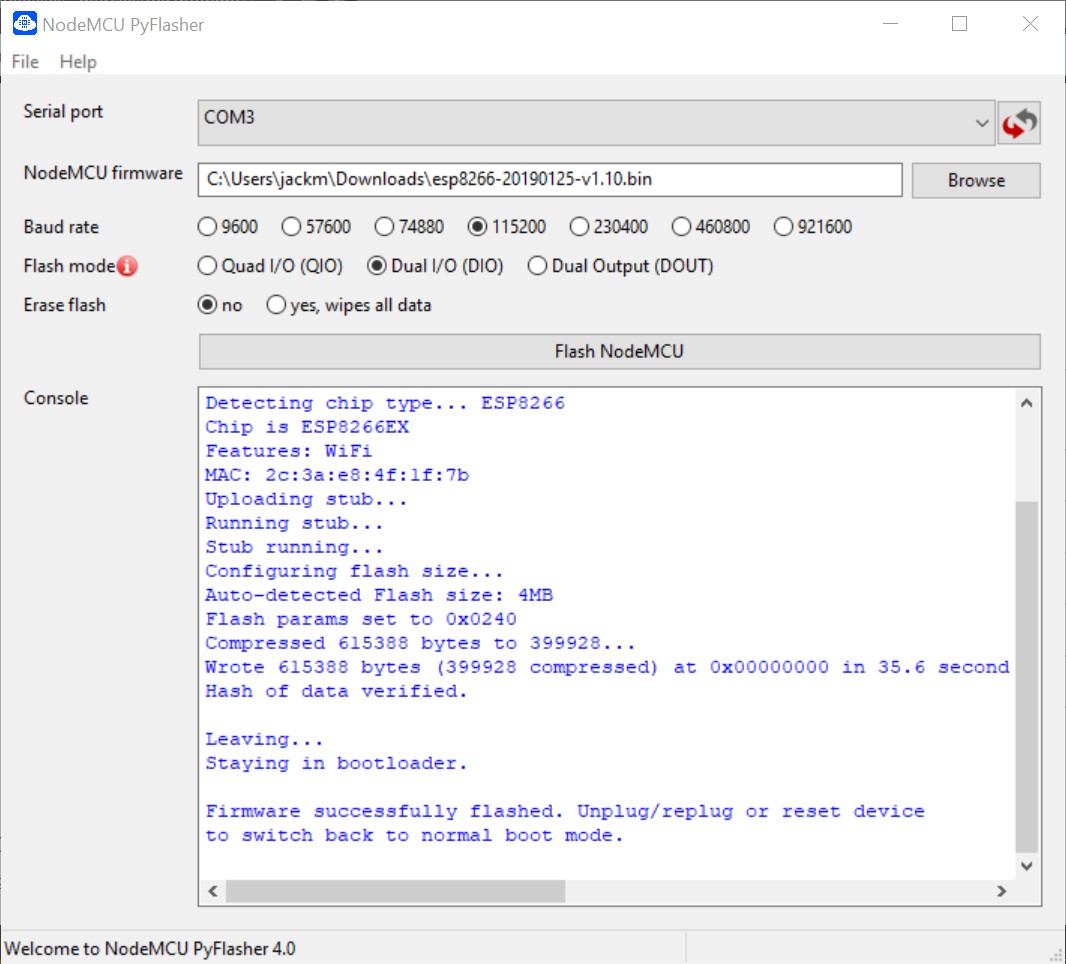
To flash the MicroPython firmware onto the NodeMCU, you can choose following technique.

1. **NodeMCU PyFlasher**

A self-contained NodeMCU flasher with GUI that utilizes esptool.py and wxPython (Python GUI library), developed by Marcel Stör. Use the latest release, which is available for Windows (.exe), macOS (.dmg) and anything that runs Python.

Once you’ve plugged your NodeMCU, you should now be able to select the COM port that it is connected on. If you don’t see it on this list, you may need to download the USB-Serial driver. Once the driver is active, you will able to determine the COM port number by either selecting it in the drop-down on NodeMCU PyFlasher or in your device manager (on Windows).

Now just select your downloaded MicroPython firmware .bin file from the download location, select a baud rate of 115200 (default for NodeMCU), using a dual I/O flash mode and click Flash NodeMCU. After the process completes, you should see a screen like below.



The window you'll see once you've successfully completed the firmware download.

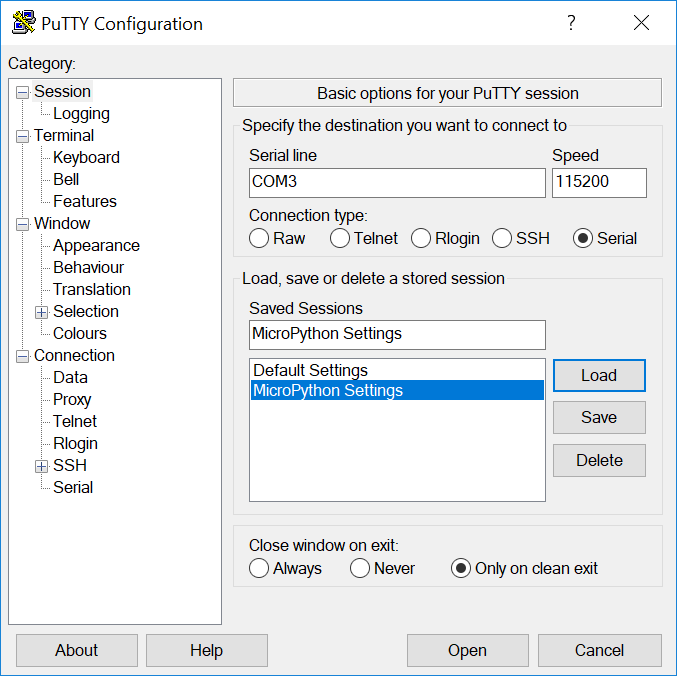
1. **Access the MicroPython REPL Prompt**

REPL (Read Evaluate Print Loop) is the name given to the interactive MicroPython prompt that you can access on the NodeMCU. Using the REPL greatly simplifies and speeds up testing out your code and running commands.

While you can access the REPL over Wi-Fi (WebREPL), this experiment will only go into how to access the REPL over a wired connection through the UART serial port.

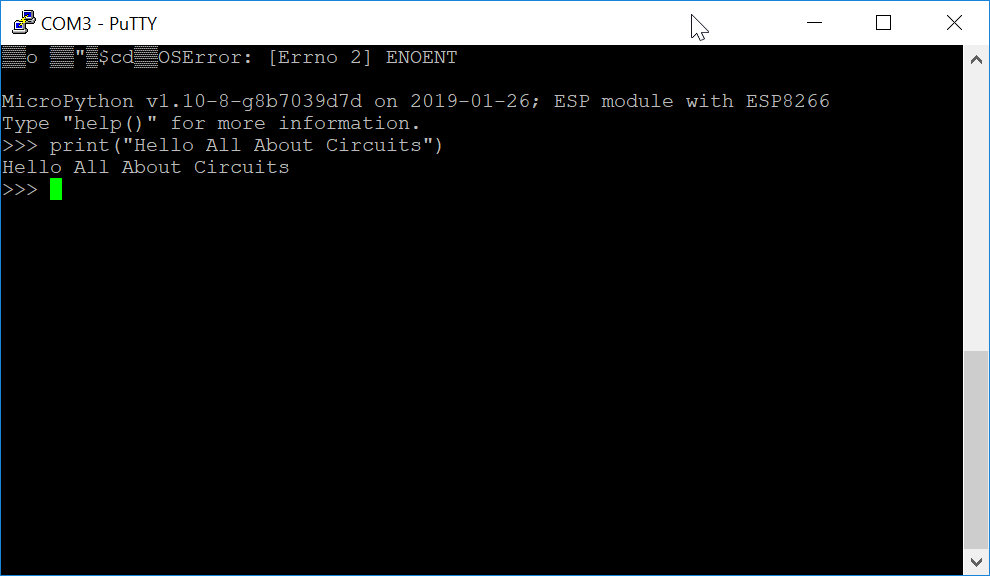
To access the prompt over USB-serial, you will need to use a terminal emulator program. PuTTY is an open source SSH and telnet client which you can use as a terminal emulator program to access your NodeMCU REPL, otherwise, there is TeraTerm on Windows, built-in screen program on Mac and Linux has picocom and minicom. There are many other terminal programs that will work, so choose your favorite.

Once you have opened PuTTY for the first time, you will need to set up the settings configuration to access the NodeMCU. First, we select a Serial connection type. Then, from step 3, we now know the COM port number that the NodeMCU board is on (COM3 for this example). From there, we can set a baud rate of 115200 and we are ready to connect. The settings can be saved as well for when you use it in the future. The settings for this example can be seen below.



The PuTTY configuration settings.

Once you connect you will be presented with a screen like below and can test out your code!



Once you're connected you'll be greeted with this screen.

1. **Make the LED Turn On**

##### Since the NodeMCU 12E has on on-board LED attached to GPIO2, you can turn it on and off using the following code:

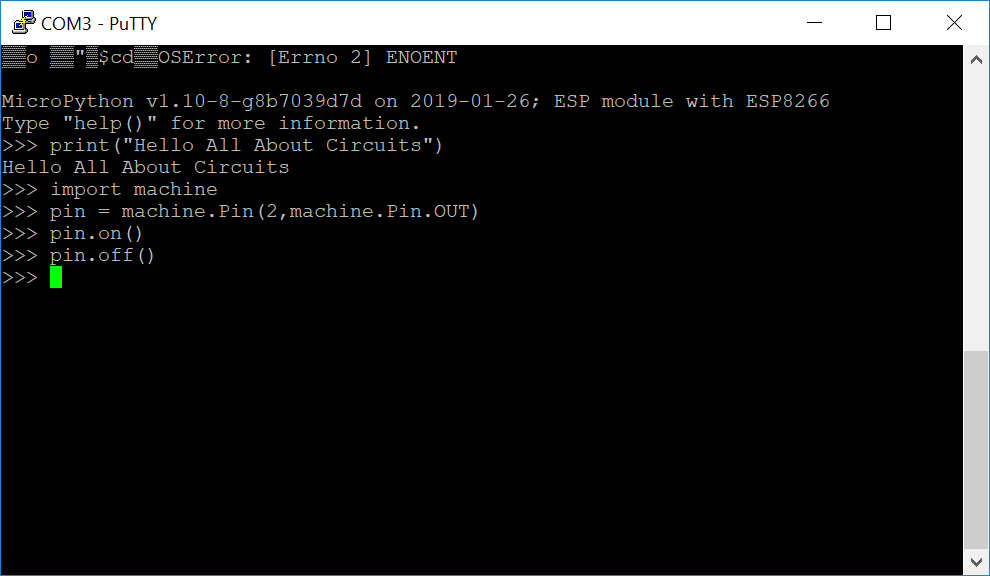
>>> **import** machine

>>> pin = machine.Pin(2, machine.Pin.OUT)

>>> pin.on()

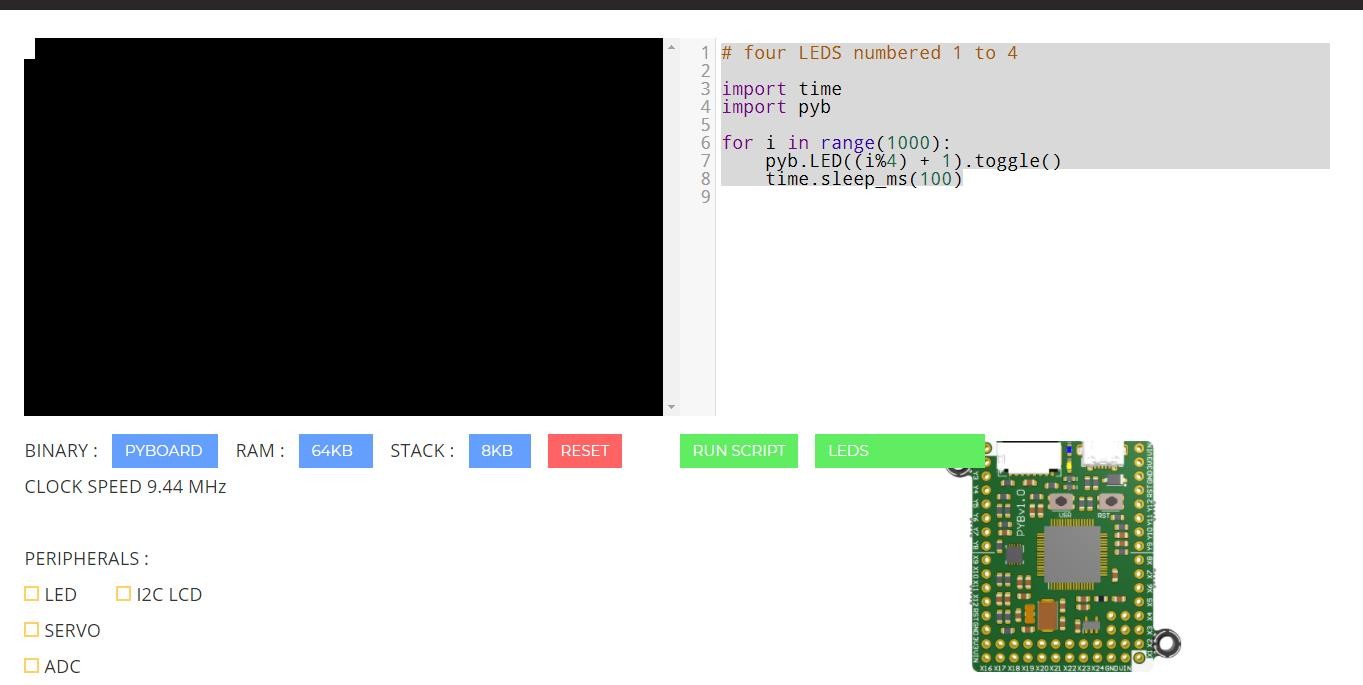
>>> pin.off()

the pin.on() may turn the LED off and pin.off() may turn the LED on (or vice versa) depending on how the LED is wired on your board.



##### The code to make the LED connected to the NodeMCU flash.

**--- Simulation of LED Blinking on https://micropython.org/unicorn/ ------**



Conclusion:

**Quiz:**( Write your answers )

1. How do you configure NodeMCU for MicroPython, and what are the essential steps involved?
2. What are GPIO pins, and how are they used to send digital data in MicroPython?

### Experiment No: 09

**Connect Digital/Analog I/O module with NodeMCU and write program to display temperature in MicroPython.**

**Competency and Practical Skills:** MicroPython understanding of NodeMCU.

**Relevant CO:** CO1, CO2, CO3,CO4

**Objectives:** To become familiar with I/O of NodeMCU. Configure NodeMCU for MicroPython.

**Equipment/Instruments:** NodeMCU 12E ESP32/8266 Board ,DHT11 Sensor,USB to Micro USB cable, Software- PuTTY, PyFlasher, 10k Ohm resistor,Breadboard,Jumper wires.

### Theory:

Micropython:

MicroPython is an implementation of Python 3, targeted for embedded systems and microcontrollers. MicroPython and regular Python are very similar, so if you already know how to write programs in Python, you already know how to program in MicroPython.

Micropython includes a small subset of the Python standard library, so not every module from the standard library is available. However, MicroPython does include modules to access low-level hardware, meaning there are modules to easily interact with GPIOs on microcontrollers.

MicroPython also comes with an interactive REPL (Read-Evaluate-Print Loop), which is an often overlooked amazing feature of MicroPython. The REPL allows you to connect to a microcontroller, execute code quickly without the need to compile or upload code. Which gives immediate feedback on whether your program is working as intended.

ESP32/ESP8266:

The ESP32 and ESP8266 are low-cost Wi-Fi modules, perfect for DIY Internet of Things (IoT) projects.Both come with general purpose input/output pins (GPIOs) and support a variety of protocols, including SPI, I2C, UART, and more. The most attractive part of the ESP range is that they come with wireless networking, separating them from their Arduino microcontroller counterparts. All in all, the ESP series allows you to easily control/monitor devices remotely using Wi-Fi for a very low price.

For this Experiment, we’ll be using the NodeMCU 12-E. We’ll be referencing this board and its instructions for this particular Experiment.



The NodeMCU 12E ESP8266 Board.

**DHT11/22:**

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity.



**DHT11 and DHT22**

These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.

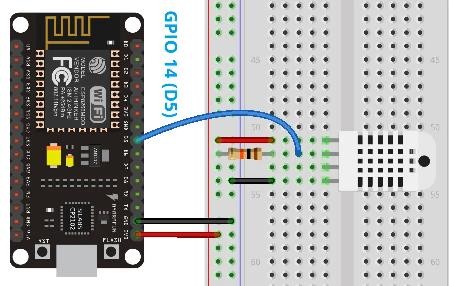
**DHT Pinout:**

DHT sensors have four pins as shown in the following figure. However, if you get your DHT sensor in a breakout board, it comes with only three pins and with an internal pull-up resistor on pin 2.

The following table shows the DHT22 and DHT11 pinout. When the sensor is facing you, pin numbering starts at 1 from left to right.

|  |  |  |
| --- | --- | --- |
|  | **DHP Pin** | **Connect to** |
| 1 | 3.3V |
| 2 | Any digital GPIO; also connect a 10k Ohm pull-up resistor |
| 3 | Don’t connect |
| 4 | GND |

### Schematic: ESP8266 with DHT11/DHT22



**Code:**

Open a new file in your MicroPython IDE (like uPyCraft IDE or Thonny IDE) and copy the following code.

from machine import Pin from time import sleep

import dht

sensor = dht.DHT22(Pin(14)) #sensor = dht.DHT11(Pin(14))

while True: try:

sleep(2) sensor.measure()

temp = sensor.temperature() hum = sensor.humidity() temp\_f = temp \* (9/5) + 32.0

print('Temperature: %3.1f C' %temp) print('Temperature: %3.1f F' %temp\_f) print('Humidity: %3.1f %%' %hum)

except OSError as e: print('Failed to read sensor.')

**How this Code Works**

1. Import the Pin class from the machine module to define pins, import the sleep method from the time module to add delays to your code, and finally import the dht module to import the functions to read from the DHT sensors.

from machine import Pin from time import sleep import dht

1. Define a dht object called sensor on the specified data pin. In this case, we’re connecting the data pin to GPIO 14. Use the following command if you’re using a DHT22 sensor:

sensor = dht.DHT22(Pin(14))

1. In the while loop, we use try and except statements. In the try statement we try to get temperature and humidity values.

In the try statement, first add a delay of two seconds because the DHT22 maximum sampling rate is two seconds. In case of the DHT11, it is one second.

sleep(2)

1. Before requesting temperature and humidity, you need to use the measure() method on the sensor object.

sensor.measure() Then, read the temperature with sensor.temperature() and the humidity with sensor.humidity(). Save those readings on the temp and hum variables.

temp = sensor.temperature() hum = sensor.humidity()

1. The following command converts the temperature to Fahrenheit degrees.

temp\_f = temp \* (9/5) + 32.0

Finally, print all the readings on the MicroPython shell using the print() function:

print('Temperature: %3.1f C' %temp) print('Temperature: %3.1f F' %temp\_f) print('Humidity: %3.1f %%' %hum)

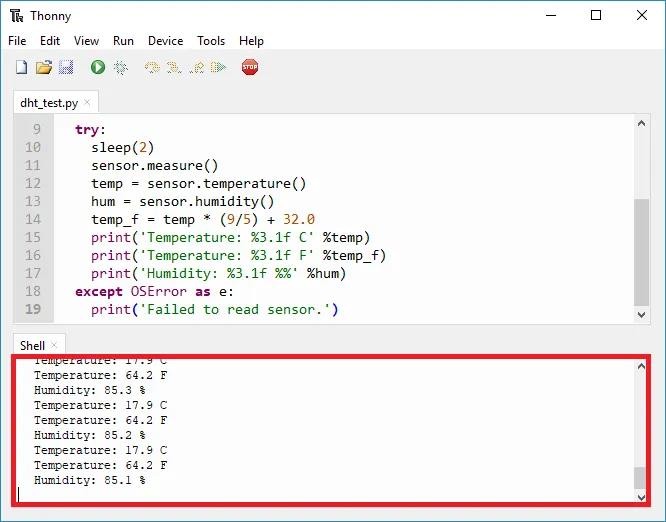
1. In case there is an error getting the readings, the except statement runs and an error message is printed:

except OSError as e: print('Failed to read sensor.')

**Demonstration:**

After copying the code and making any necessary changes, you can upload the code to your ESP32 or ESP8266.

If you’re using Thonny IDE, you just need to go to Device > Upload current script as main script. After uploading the code, press the ESP on-board RESET button ,New temperature and humidity readings should be displayed on the MicroPython Shell, every two seconds.



Conclusion:

**Quiz:**( Write your answers in separate page and attached the page with this manual)

1. What is the purpose of connecting a digital/analog I/O module with NodeMCU?
2. What type of sensors can be used to measure temperature in this project?

### Experiment No: 10

**Connect NodeMCU 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 or any other cloud service)**

**Competency and Practical Skills:** MicroPython understanding of NodeMCU. Cloud services

Relevant CO: CO3,CO4

**Objectives:** To become familiar with I/O of NodeMCU. Configure NodeMCU for MicroPython.

**Equipment/Instruments:** NodeMCU 12E ESP32/8266 Board ,DHT11 Sensor,USB to Micro USB cable, Software- PuTTY, PyFlasher, 10k Ohm resistor, Breadboard, Jumper wires

### Theory:

Micropython:

MicroPython is an implementation of Python 3, targeted for embedded systems and microcontrollers. MicroPython and regular Python are very similar, so if you already know how to write programs in Python, you already know how to program in MicroPython.

Micropython includes a small subset of the Python standard library, so not every module from the standard library is available. However, MicroPython does include modules to access low-level hardware, meaning there are modules to easily interact with GPIOs on microcontrollers.

MicroPython also comes with an interactive REPL (Read-Evaluate-Print Loop), which is an often overlooked amazing feature of MicroPython. The REPL allows you to connect to a microcontroller, execute code quickly without the need to compile or upload code. Which gives immediate feedback on whether your program is working as intended.

ESP32/ESP8266:

The ESP32 and ESP8266 are low-cost Wi-Fi modules, perfect for DIY Internet of Things (IoT) projects.Both come with general purpose input/output pins (GPIOs) and support a variety of protocols, including SPI, I2C, UART, and more. The most attractive part of the ESP range is that they come with wireless networking, separating them from their Arduino microcontroller counterparts. All in all, the ESP series allows you to easily control/monitor devices remotely using Wi-Fi for a very low price.

For this Experiment, we’ll be using the NodeMCU 12-E. We’ll be referencing this board and its instructions for this particular Experiment.



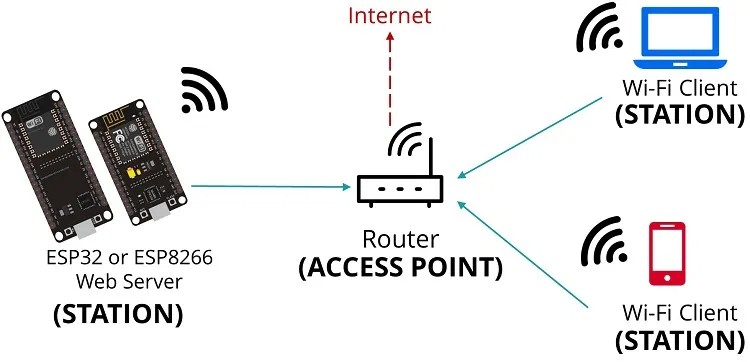
The NodeMCU 12E ESP8266 Board.

**Demonstration:**

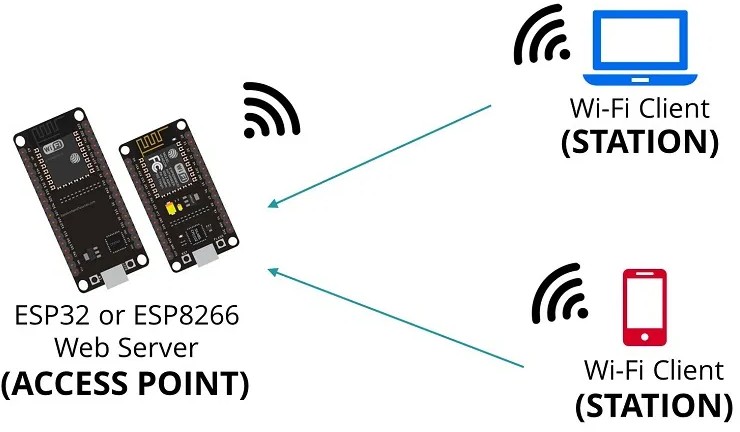
**ESP32/ESP8266 Station and Access Point:**

In most of our web server projects with MicroPython, we connect the ESP32 or the ESP8266 to a wireless router. In this configuration, we can access the ESP board through the local network.

In this scenario, the router acts as an access point and the ESP boards are set as a station. So, you need to be connected to your router (local network) to control the ESP32 or ESP8266.



In some cases, this might not be the best configuration (when you don’t have a router nearby). But if you set the ESP boards as an access point (hotspot), you can be connected to them using any device with Wi-Fi capabilities without the need to connect to your router.

Basically, when you set the ESP32 or ESP8266 as an access point you create its own Wi-Fi network and nearby Wi-Fi devices (stations) can connect to it (like your smartphone or your computer).

**Logging to the Cloud:**

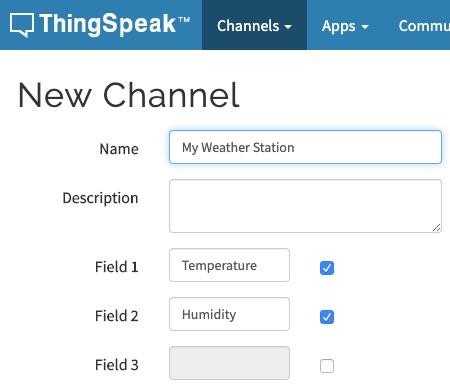
The next task is to set up an account at thingspeak.com. This is a cloud service that stores data series,

specifically designed for Internet of Things devices. The free tier offered by this service is more than enough for the needs of this experiment.



Once you complete the account set up and verify your email address, click the "Channels" dropdown in the top navigation bar, and then select "My Channels".

Click the button to create a channel where you'll upload your sensor data. In ThingSpeak, a channel is a data structure that can hold timed series. Each channel can have up to eight fields, but for this exercise you need just two to hold the temperature and humidity series. See below how I configured my channel:



The remaining options in the New Channel screen can be left alone. Once you added the two fields, click the "Save Channel" button at the bottom.

Now you should be in the new channel's page. The channel has its own navigation bar, and you are seeing the "Private View" page, where there are still empty charts for the two fields that you created in this channel. Click on "API Keys" on this navigation bar to access the page that has the information on how to upload data to the channel.

The API Keys page shows the two keys that were assigned to your channel, one for writing data to it and another for reading. On the right sidebar there is a section titled "API Requests", which shows a few example HTTP requests that you can send for different tasks. The one that is interesting at this point is the "Upload a Channel Feed", which shows the GET request that needs to be sent to write some data into the channel. It should be similar to this, but with a different API key:

GET https://api.thingspeak.com/update?api\_key=IQBL4LHQIQ84YH57&field1=0

ThingSpeak uses a GET request for data submission, which is not very conventional because requests of this type cannot carry a payload. In general a POST request is used when submitting data to a web service.

But if GET requests cannot carry data with them, how is the data submitted? Look at the above URL, in particular to the part that follows after the ? sign. Let me repeat that part here with spaces added for clarity:

? api\_key=IQBL4LHQIQ84YH57 & field1=0

The HTTP protocol calls this the [query string](https://en.wikipedia.org/wiki/Query_string) part of the URL. It starts with a question mark, and is followed by one or more parameter assignments. If multiple parameters are included, they are separated with an ampersand sign. The thingspeak URL uses a parameter called api\_key for the write API key, which is what identifies your channel. Then a second parameter called field1 passes a value for the first field in the channel. So as you see, uploading data to ThingSpeak implies generating this URL that includes parameters and then sending a GET request with it. Since your channel has two fields, the URL is going to have one more parameter in it:

GET

https://api.thingspeak.com/update?api\_key=IQBL4LHQIQ84YH57&field1={t}&field2={h}

In this URL, I used {t} and {h} to represent the values of temperature and humidity read from the sensor. These will have to be incorporated into the URL before the request is made.

Select the upload URL shown in the API Keys page of your channel and paste it on your config.py file. Then edit it so that it matches the following format:

WIFI\_SSID = 'your SSID'

WIFI\_PASSWORD = 'your Wi-Fi password' LED\_PIN = 2 # D4

LED2\_PIN = 16 # D0 DEBUG\_PIN = 14 # D5 DHT22\_PIN = 4 # D2 FAHRENHEIT = False WEBHOOK\_URL =

'https://api.thingspeak.com/update?api\_key=IQBL4LHQIQ84YH57&field1={temperature}&field 2={humidity}'

When you do this, make sure that the value of the api\_key parameter is your channel's write key, not mine. Also make sure you write the values for the temperature and humidity exactly

as {temperature} and {humidity}, as these will be replaced with the actual values before the request is sent to ThingSpeak.

main.py for the temperature and humidity cloud uploader application:

import dht import machine import network import sys import time import urequests

import config

def connect\_wifi():

ap\_if = network.WLAN(network.AP\_IF) ap\_if.active(False)

sta\_if = network.WLAN(network.STA\_IF) if not sta\_if.isconnected():

print('Connecting to WiFi...') sta\_if.active(True)

sta\_if.connect(config.WIFI\_SSID, config.WIFI\_PASSWORD) while not sta\_if.isconnected():

time.sleep(1)

print('Network config:', sta\_if.ifconfig())

def show\_error():

led = machine.Pin(config.LED\_PIN, machine.Pin.OUT) led2 = machine.Pin(config.LED2\_PIN, machine.Pin.OUT) for i in range(3):

led.on() led2.off() time.sleep(0.5) led.off()

led2.on() time.sleep(0.5)

led.on()

led2.on()

def is\_debug():

debug = machine.Pin(config.DEBUG\_PIN, machine.Pin.IN, machine.Pin.PULL\_UP) if debug.value() == 0:

print('Debug mode detected.') return True

return False

def get\_temperature\_and\_humidity():

dht22 = dht.DHT22(machine.Pin(config.DHT22\_PIN)) dht22.measure()

temperature = dht22.temperature() if config.FAHRENHEIT:

temperature = temperature \* 9 / 5 + 32 return temperature, dht22.humidity()

def log\_data(temperature, humidity): print('Invoking log webhook')

url = config.WEBHOOK\_URL.format(temperature=temperature,

humidity=humidity)

response = urequests.get(url) if response.status\_code < 400:

print('Webhook invoked') else:

print('Webhook failed')

raise RuntimeError('Webhook failed')

def run():

try:

connect\_wifi()

temperature, humidity = get\_temperature\_and\_humidity() log\_data(temperature, humidity)

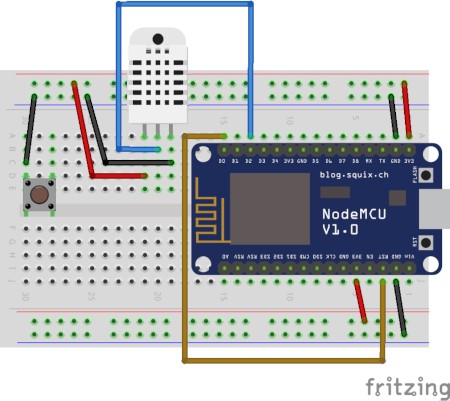
except Exception as exc: sys.print\_exception(exc) show\_error()

if not is\_debug(): machine.deepsleep()

run()

If you upload main.py and config.py to your microcontroller, each time you power it or reset it

a record of the current temperature and humidity will be uploaded to your channel on thingspeak.com.



**Schematic**

**Output:**



Conclusion:

**Quiz:**( Write your answers)

* 1. What is the purpose of connecting a digital/analog I/O module with NodeMCU?
  2. What type of sensors can be used to measure temperature in this project?