**NAME: NASHRA GHAFFAR**

**ROLL NO: CT-20032**

**AI LAB MANUAL**

**Lab 1**

**1. What is an expression?**

An expression is a combination of values, variables, operators, and function calls that can be evaluated to produce a result.

**2. What is a syntax error?**

A syntax error occurs when the code violates the rules of the programming language, making it unable to be parsed or executed.

**3. What is the result of this expression: “\*” \* 10.**

The result of the expression "*"10 is a string containing ten consecutive asterisks: "*\*\*\*\*\*\*\*\*".

**4. What is a variable?**

A variable is a named storage location in a program that holds a value or reference to a value.

**5. What are the primitive built-in types in Python?**

The primitive built-in types in Python are:

- Integer (`int`): Whole numbers without decimal points.

- Float (`float`): Real numbers with decimal points.

- String (`str`): Sequence of characters.

- Boolean (`bool`): Represents the values `True` or `False`.

- NoneType (`None`): Represents the absence of a value.

**6. When should we use “”” (triple quotes) to define strings?**

We use triple quotes ("""...""") to define multi-line strings or strings that span across multiple lines in Python.

**7. Assuming (name = “John Smith”). What does name[1] return?**

1. **name[1]** returns the character at the index 1 of the string "John Smith", which is "o".
2. **name[-2]** returns the character at the second-to-last index of the string "John Smith", which is "t".
3. **name[1:-1]** returns a substring of "John Smith" starting from index 1 (inclusive) to the second-to-last index (exclusive), which is "ohn Smit".
4. To get the length of **name**, you can use the **len()** function like this: **len(name)**.

**8. Given (name = “john smith”), what will name.title() return?**

`**name.title()** will return "John Smith" by capitalizing the first letter of each word in the string.

**9. What does name.strip() do?**

**name.strip()** removes any leading and trailing whitespaces from the string "John Smith".

**10. What will name.find(“Smith”) return?**

**name.find("Smith")** will return the index of the substring "Smith" in the string "John Smith". In this case, it will return 5

.**11. What will be the value of name after we call name.replace(“j”, “k”)?**

After calling **name.replace("j", "k")**, the value of **name** will still be "John Smith" as Python's string methods return a new string and do not modify the original string in-place.

**12. How can we check to see if name contains “John”?**

You can check if "John" is present in the string `name` using the `in` keyword like this: `if "John" in name:`.

**13. What are the 3 types of numbers in Python?**

The three types of numbers in Python are:

- Integer (`int`): Whole numbers without decimal points.

- Float (`float`): Real numbers with decimal points.

- Complex (`complex`): Numbers in the form `a + bj`, where `a` and `b` are real numbers, and `j` represents the imaginary unit.

**14. What is the difference between 10 / 3 and 10 // 3?**

`10 / 3` will give the floating-point division result, which is `3.3333333333333335`.

`10 // 3` will give the integer division result, which is `3`. It truncates the fractional part and returns the integer result.

**15. What is the result of 10 \*\* 3?**

The expression `10 \*\* 3` will give `1000`, which is 10 raised to the power of 3.

**16. Given (x = 1), what will be the value of x after we run (x += 2)?**

After running `(x += 2)`, the value of `x` will be `3`. It's a shorthand for `x = x + 2`.

**17. How can we round a number?**

You can round a number using the built-in `round()` function. For example, `rounded\_num = round(3.14159)` will give `3`.

**18. What is the result of float(1)?**

The expression `float(1)` will give `1.0`. It converts the integer `1` into a floating-point number.

**19. What is the result of 10 == “10”?**

The expression `10 == "10"` will return `False`. It checks if the value of `10` (an integer) is equal to the value of `"10"` (a string). Since the types are different, the result is `False`.

**20. What is the result of “bag” > “apple”?**

The expression `"bag" > "apple"` will return `True`. Python compares strings based on their lexicographic order, and in this case, "bag" comes after "apple" in the dictionary.

**21. What is the result of not(True or False)?**

The expression `not(True or False)` will return `False`. The `not` operator negates the value of the expression, so if the expression is `True`, `not` makes it `False`, and vice versa.

**22. What does range(1, 10, 2) return?**

`range(1, 10, 2)` returns an iterable object that represents the sequence of numbers starting from 1 up to (but not including) 10, with a step of 2. So, it will return the numbers 1, 3, 5, 7, and 9.

**23. Write a function that returns the maximum of two numbers.**

def max\_of\_two(a, b):

return max(a, b)

**24. Write a function called check\_num that takes a number.**

def check\_num(number):

if number % 3 == 0 and number % 5 == 0:

return "Divisible by both"

elif number % 3 == 0:

return "Divisible by 3"

elif number % 5 == 0:

return "Divisible by 5"

**25. Write a function called showNumbers that takes a parameter called limit. It should print all the numbers between 0 and limit.**

def showNumbers(limit):

for i in range(limit + 1):

print(i)

**26. Write a Python program which accepts the radius of a circle from the user and computes the area of a circle, triangle, square & rectangle.**

import math

def calculate\_area(shape, radius):

if shape == "circle":

return math.pi \* radius \*\* 2

elif shape == "triangle":

return (math.sqrt(3) / 4) \* radius \*\* 2

elif shape == "square":

return radius \*\* 2

elif shape == "rectangle":

return radius \* 2

shape = input("Enter the shape (circle, triangle, square, rectangle): ")

radius = float(input("Enter the radius: "))

area = calculate\_area(shape, radius)

print(f"The area of {shape} with radius {radius} is: {area:.2f}")

**27. Write a function that returns the sum of multiples of 3 and 5 between 0 and limit (parameter).**

def sum\_multiples(limit):

result = 0

for num in range(limit + 1):

if num % 3 == 0 or num % 5 == 0:

result += num

return result

**28. Write a function called show\_stars(rows). If rows is 5, it should print the following:**

def show\_stars(rows):

for i in range(1, rows + 1):

print("\*" \* i)

**29. Write a function that prints all the prime numbers between 0 and limit where limit is a parameter.**

def is\_prime(number):

if number < 2:

return False

for i in range(2, int(number \*\* 0.5) + 1):

if number % i == 0:

return False

return True

def print\_primes(limit):

for num in range(limit + 1):

if is\_prime(num):

print(num)

**30. Write a Python program to display the first and last colors from the following list.**

**color\_list = ["Red", "Green", "White", "Black"]**

color\_list = ["Red", "Green", "White", "Black"]

first\_color = color\_list[0]

last\_color = color\_list[-1]

print(f"First color: {first\_color}, Last color: {last\_color}")

**31. Write a Python program which accepts the user's first and last name and prints them in reverse order with a space between them.**

first\_name = input("Enter your first name: ")

last\_name = input("Enter your last name: ")

reversed\_name = last\_name + " " + first\_name

print("Reversed name:", reversed\_name)

**32. Write a Python program to find whether a given number (accept from the user) is even or odd, and print out an appropriate message to the user.**

def is\_even\_or\_odd(number):

if number % 2 == 0:

return "Even"

else:

return "Odd"

num = int(input("Enter a number: "))

result = is\_even\_or\_odd(num)

print(f"{num} is {result}.")

**33. Write a Python program that accepts an integer (n) and computes the value of n + nn + nnn.**

def compute\_value(n):

nn = int(str(n) \* 2)

nnn = int(str(n) \* 3)

return n + nn + nnn

num = int(input("Enter an integer: "))

result = compute\_value(num)

print(f"The result is: {result}")

**34. Write a Python program to test whether a passed letter is a vowel or not.**

def is\_vowel(letter):

vowels = "AEIOUaeiou"

return letter in vowels

letter = input("Enter a letter: ")

if is\_vowel(letter):

print(f"{letter} is a vowel.")

else:

print(f"{letter} is not a vowel.")

**35. Write a Python program to sum three given integers. However, if two values are equal, the sum will be zero.**

def sum\_with\_condition(a, b, c):

if a == b or b == c or a == c:

return 0

return a + b + c

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

num3 = int(input("Enter the third number: "))

result = sum\_with\_condition(num1, num2, num3)

print(f"The sum is: {result}")

**36. Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.**

def check\_values(a, b):

return a == b or abs(a - b) == 5 or a + b == 5

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = check\_values(num1, num2)

print(result)

**37. Write a Python program to solve (x + y) ^ (z).**

x = 4

y = 3

z = 2

result = (x + y) \*\* z

print(f"({x} + {y}) ^ {z} = {result}")

**LAB 2**

Q1: Define the following terms:

**Regular Graph:** A regular graph is a type of graph where all its vertices have the same degree, meaning each vertex has an equal number of edges incident to it.

**Null Graph**: The null graph is the simplest type of graph, containing no vertices and no edges. It is often denoted as "N" or "Φ."

**Trivial Graph:** The trivial graph is another simple graph that consists of a single vertex without any edges. It is sometimes referred to as a singleton graph.

**Simple Graph**: A simple graph is an undirected graph that does not contain any self-loops (edges from a vertex to itself) or parallel edges (multiple edges between the same pair of vertices).

**Connected Graph**: A connected graph is a graph in which there is a path between every pair of vertices. In other words, for any two vertices in the graph, there exists a sequence of edges that connect them.

**Disconnected Graph:** A disconnected graph is a graph that is not connected. It consists of two or more connected components, where there is no path between vertices in different components.

**Complete Graph:** A complete graph is a simple graph in which each vertex is directly connected to every other vertex by a unique edge. It is denoted as "Kn," where "n" represents the number of vertices.

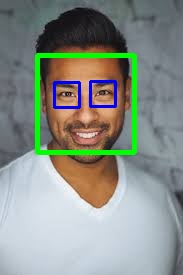
**Cyclic Graph:** A cyclic graph is a graph that contains at least one cycle, which is a closed path that starts and ends at the same vertex, passing through a series of distinct vertices and edges.

**Degree of vertex**: The degree of a vertex in a graph refers to the number of edges incident to that vertex. For undirected graphs, it counts both incoming and outgoing edges.

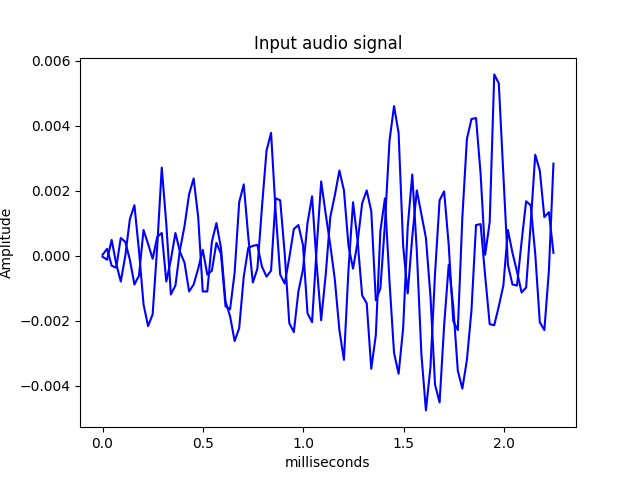
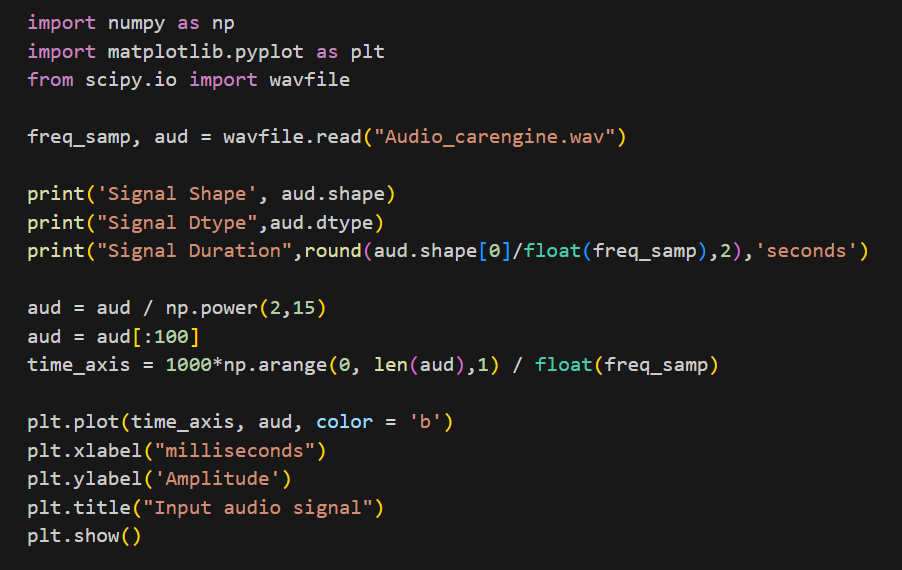
**Loop**: A loop is an edge in a graph that connects a vertex to itself. In other words, it is an edge with both endpoints being the same vertex

**Parallel Edges:** Parallel edges refer to multiple edges between the same pair of vertices in a graph. In simple graphs, parallel edges are not allowed, but they can exist in multigraphs (graphs that permit multiple edges between vertices).

**Lab 7**

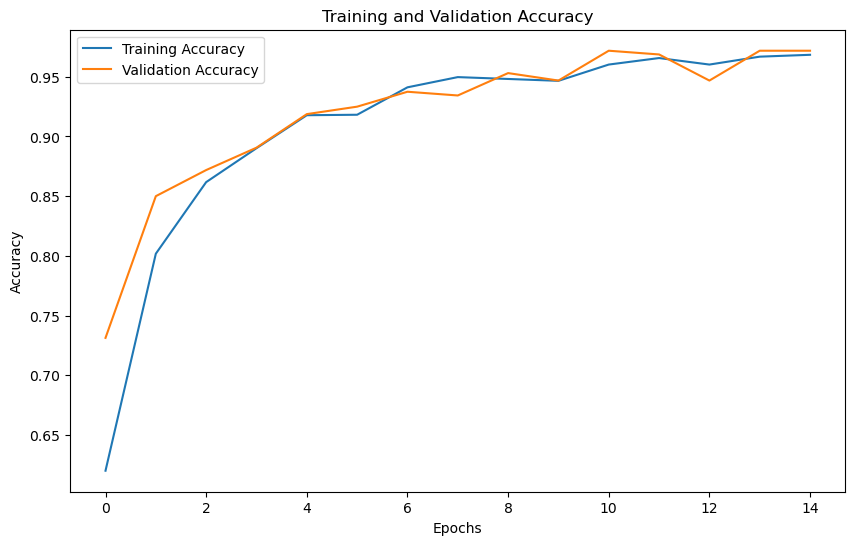
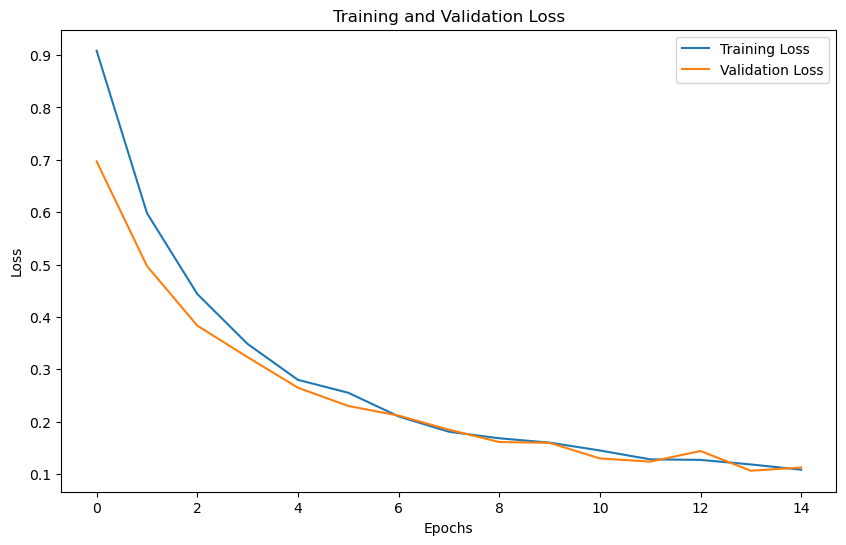


**Lab 8**



**Lab 9**

**Deep Neural Network Validation and Loss Graphs:**

** **

**Model Comparison:**

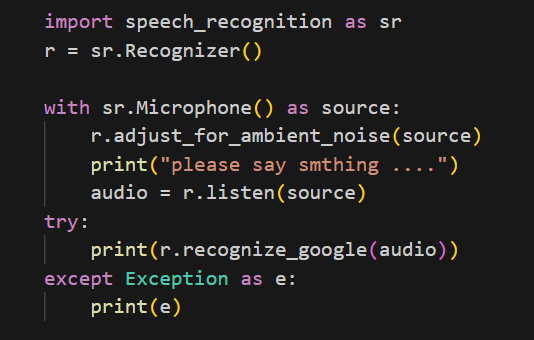
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **Accuracy** | **Confusion Matrix** | **F1 Score** |
| **Deep Neural Network** | 0.975 | 0.964 | 0.972 | [[ 68 6 0]  [ 0 109 2]  [ 0 1 144]] | 0.969 |
| **Random Forest Classifier** | 0.977 | 0.977 | 0.977 |  | 0.974 |
| **Support Vector Machine** | 0.96 | 0.99 | 0.97 | [[142 4 3]  [ 1 213 1]  [ 0 6 216]] | 0.97 |

**Lab 10**

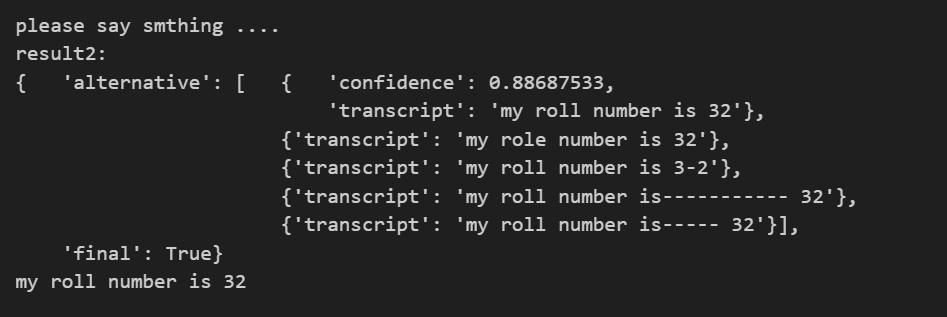
**Model Comparison:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **Accuracy** | **Confusion Matrix** | **F1 Score** |
| **Deep Neural Network** | 0.975 | 0.964 | 0.972 | [[ 68 6 0]  [ 0 109 2]  [ 0 1 144]] | 0.969 |
| **Random Forest Classifier** | 0.977 | 0.977 | 0.977 |  | 0.974 |

**Lab 11**

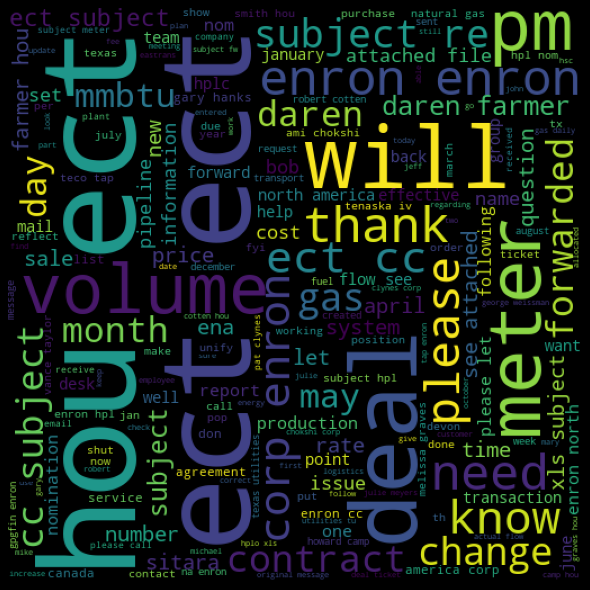
****

**Output:**

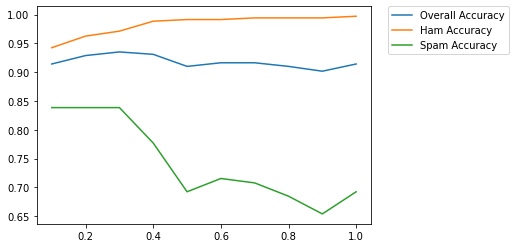


**Lab 12**

**Word Cloud:**

****

**Evaluation Graph:**



Accuracy for ham without stop words:

0.9971264367816092

Accuracy for spam without stop words:

0.6923076923076923

Overall Accuracy without stop words:

0.9142259414225942