Final_Assignment

Question 1.1: Write the Answer to these questions.

1. What is the difference between static and dynamic variables in Python?

- **Static Variables**: These variables are bound to the class and not the instances. They are typically defined outside of methods but inside the class and can be shared among all instances of the class. Static variables are initialized only once and do not change unless explicitly modified.

Example:

```
class MyClass:

    static_var = 10  # Static variable shared among all instances



obj1 = MyClass()

obj2 = MyClass()

obj1.static_var = 20  # Changes will reflect on all instances

print(obj2.static_var)  # Output: 20
```

- **Dynamic Variables**: These variables are defined within methods and are tied to the instance of the class. Each instance has its own copy of dynamic variables, and they can change during runtime.

Example:

```
class MyClass:

    def __init__(self, value):

        self.dynamic_var = value  # Instance-specific variable



obj1 = MyClass(10)

obj2 = MyClass(20)

obj1.dynamic_var = 30  # Only affects obj1

print(obj2.dynamic_var)  # Output: 20
```

2. Explain the purpose of `pop()`, `popitem()`, and `clear()` in a dictionary with suitable examples.

- **`pop(key)`**: Removes and returns the value of the specified key. If the key doesn't exist, it raises a `KeyError`.

Example:
```
my_dict = {'a': 1, 'b': 2}

value = my_dict.pop('a')  # Removes 'a' and returns its value

print(value)  # Output: 1

print(my_dict)  # Output: {'b': 2}
```

- **`popitem()`**: Removes and returns an arbitrary (key, value) pair from the dictionary. If the dictionary is empty, it raises a `KeyError`.

Example:
```
my_dict = {'a': 1, 'b': 2}

item = my_dict.popitem()  # Removes and returns a random (key, value) pair

print(item)  # Output: ('b', 2)
```

- **`clear()`**: Removes all items from the dictionary.

Example:
```
my_dict = {'a': 1, 'b': 2}

my_dict.clear()

print(my_dict)  # Output: {}
```

3. What do you mean by FrozenSet?

A **FrozenSet** is an immutable version of a set. Once a FrozenSet is created, it cannot be modified (no adding or removing elements). It is useful when you want to ensure the integrity of data and prevent accidental modifications. Since it is immutable, it can also be used as a key in a dictionary.

Example:
```
frozen_set = frozenset([1, 2, 3, 4])
```

```
print(frozen_set)  # Output: frozenset({1, 2, 3, 4})

# frozen_set.add(5)  # Raises AttributeError because it's immutable
```

-

4. Differentiate between mutable and immutable data types in Python and give examples of mutable and immutable data types.

- **Mutable Data Types**: Objects whose values can be changed after creation. These objects are typically stored in memory locations that can be modified.
    - Examples: `list`, `dict`, `set`

Example:

```
lst = [1, 2, 3]

lst[0] = 100  # Mutable, list can be changed

print(lst)  # Output: [100, 2, 3]
```

    -
- **Immutable Data Types**: Objects whose values cannot be changed once they are created. These objects are stored in memory locations that are read-only.
    - Examples: `int`, `str`, `tuple`, `frozenset`

Example:

```
tup = (1, 2, 3)

# tup[0] = 100  # Raises TypeError, tuple is immutable
```

    -

5. What is `__init__`? Explain with an example.

`__init__` is a special method in Python known as the **constructor**. It is automatically invoked when a new object of the class is created. It initializes the object's attributes. This method is usually used to set up an object's state by assigning values to its properties when the object is created.

Example:

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age
```

```
    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

person = Person("Alice", 30)
person.greet()  # Output: Hello, my name is Alice and I am 30 years old.
```

-

6. What is docstring in Python? Explain with an example.

A **docstring** is a string literal that appears at the beginning of a class, function, or module. It is used to document the functionality of that block of code. Docstrings are typically used to provide an explanation of what the class or function does, its parameters, and its return values.

Example:

```
 def greet(name):
    """
    This function takes a name as an argument and prints a greeting message.
    """
    print(f"Hello, {name}!")

print(greet.__doc__)  # Output: This function takes a name as an argument and prints a greeting message.
```

-

7. What are unit tests in Python?

Unit tests are used to test individual units or components of a program to ensure they work as expected. Unit testing helps ensure that your code behaves correctly and reliably over time. Python provides the `unittest` module for writing unit tests.

Example:

```
 import unittest

def add(a, b):
```

```
    return a + b


class TestAddition(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)


if __name__ == "__main__":
    unittest.main()  # Run the tests
```

-

8. What is `break`, `continue`, and `pass` in Python?

- **`break`**: Terminates the nearest enclosing loop (for, while).
- **`continue`**: Skips the current iteration of the loop and moves to the next iteration.
- **`pass`**: Does nothing. It is a placeholder used in loops, functions, or classes when you want to have no action.

Example:

```
 # break example
for i in range(5):
    if i == 3:
        break
    print(i)  # Output: 0, 1, 2


# continue example
for i in range(5):
    if i == 3:
        continue
```

```
    print(i)  # Output: 0, 1, 2, 4


# pass example

def dummy_function():

    pass  # No operation
```

-

9. What is the use of `self` in Python?

`self` is a reference to the current instance of the class. It allows you to access the instance's attributes and methods. By using `self`, you can refer to the object's data and make it available to the methods within the class.

Example:

```
 class Person:

    def __init__(self, name):

        self.name = name  # self refers to the current instance



    def greet(self):

        print(f"Hello, {self.name}!")



person = Person("Alice")

person.greet()  # Output: Hello, Alice!
```

-

10. What are global, protected, and private attributes in Python?

- **Global attributes**: Defined outside of any class or function, accessible from anywhere in the code.
- **Protected attributes**: Defined within a class, conventionally marked with a single underscore (`_`). It is not meant to be accessed directly outside the class, but it is not strictly private.

- **Private attributes**: Defined within a class and marked with double underscores (`__`). They are not accessible directly from outside the class.

Example:

```
class Example:

    global_attr = "Global"  # Global attribute



    def __init__(self):

        self._protected_attr = "Protected"  # Protected attribute

        self.__private_attr = "Private"  # Private attribute



obj = Example()

print(obj.global_attr)  # Accessible globally

print(obj._protected_attr)  # Conventionally protected, but still accessible

# print(obj.__private_attr)  # Raises AttributeError, as it's private
```

11. What are modules and packages in Python?

- **Module**: A module is a single file (with a `.py` extension) that contains Python code, functions, classes, and variables. You can import a module into another Python script to reuse its functionality.

Example:

```
 # my_module.py
def greet(name):
    print(f"Hello, {name}!")

# main.py
import my_module
my_module.greet("Alice")  # Output: Hello, Alice!
```

    -

- **Package**: A package is a collection of modules in a directory, typically with an `__init__.py` file. It allows for a hierarchical organization of Python modules.

Example:

```
 my_package/
    __init__.py
```

module1.py
module2.py
You can import modules from the package:
from my_package import module1

○

12. What are lists and tuples? What is the key difference between the two?

- **Lists**: Lists are ordered, mutable collections of elements. They can be modified after creation (adding, removing, or changing elements).

Example:
lst = [1, 2, 3]
lst[1] = 4 # Mutable
print(lst) # Output: [1, 4, 3]

○

- **Tuples**: Tuples are ordered, immutable collections of elements. Once created, their values cannot be modified.

Example:
tup = (1, 2, 3)
# tup[1] = 4 # Raises TypeError
print(tup) # Output: (1, 2, 3)

○

- **Key Difference**: The main difference is that lists are mutable, meaning their elements can be changed, while tuples are immutable, and their elements cannot be altered.

13. What is an Interpreted language & dynamically typed language? Write 5 differences between them.

- **Interpreted Language**: An interpreted language is one in which code is executed line-by-line, directly by an interpreter, rather than being compiled into machine code first. Python is an interpreted language.
    - Example: Python code is executed line by line, without a compilation step.
- **Dynamically Typed Language**: In a dynamically typed language, variable types are determined at runtime rather than at compile time.

Example: In Python, you don't need to declare the type of a variable:
x = 5 # x is an integer
x = "Hello" # x is now a string

○

Differences:

1. **Interpretation**: Interpreted languages are executed line-by-line, while compiled languages are first translated into machine code before execution.
2. **Typing**: Dynamically typed languages determine variable types at runtime, whereas statically typed languages require explicit type declarations.
3. **Execution Speed**: Interpreted languages are generally slower than compiled languages because of line-by-line execution.
4. **Error Checking**: Errors in dynamically typed languages can only be caught at runtime, whereas statically typed languages catch type-related errors at compile time.
5. **Flexibility**: Dynamically typed languages are more flexible, allowing variable type changes, while statically typed languages enforce fixed types.

14. What are Dict and List comprehensions?

- **List Comprehension**: A compact way to create lists based on an existing list or iterable.

Example:
```
lst = [x**2 for x in range(5)]  # List of squares
print(lst)  # Output: [0, 1, 4, 9, 16]
```

- **Dictionary Comprehension**: A similar way to create dictionaries based on an iterable, using key-value pairs.

Example:
```
my_dict = {x: x**2 for x in range(5)}
print(my_dict)  # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

15. What are decorators in Python? Explain it with an example. Write down its use cases.

Decorators are functions that modify or extend the behavior of other functions or methods. They are commonly used for logging, access control, caching, etc. Decorators are applied using the `@` symbol above the function definition.

Example:
```
def decorator(func):
    def wrapper():
        print("Before function execution")
        func()
        print("After function execution")
    return wrapper

@decorator
def greet():
```

```
    print("Hello!")

greet()  # Output: Before function execution
         #         Hello!
         #         After function execution
```

-

Use Cases:

- **Logging**: Log function calls.
- **Authorization**: Check if the user has the right permissions.
- **Caching**: Cache the results of function calls.
- **Validation**: Validate input before passing it to a function.

16. How is memory managed in Python?

Python uses **automatic memory management**, which involves:

- **Garbage Collection**: Python automatically frees up memory by cleaning up objects that are no longer in use. This is done by reference counting and a cyclic garbage collector to handle objects that refer to each other.
- **Memory Pooling**: Python internally manages memory in pools to reduce fragmentation.
- **Dynamic Typing**: Memory allocation happens dynamically at runtime.

17. What is lambda in Python? Why is it used?

A **lambda function** is an anonymous, one-line function defined using the `lambda` keyword. It can have any number of arguments but can only have one expression. It is useful when you need a quick, simple function without having to define it using `def`.

Example:

```
 square = lambda x: x**2
print(square(5))  # Output: 25
```

-

Use Case: Lambda functions are commonly used in functions like `map()`, `filter()`, and `sorted()` where a simple function is needed for a short task.

18. Explain `split()` and `join()` functions in Python.

- **`split()`**: Splits a string into a list of substrings based on a delimiter.

Example:

```
 text = "apple,orange,banana"
fruits = text.split(",")
```

```
print(fruits)  # Output: ['apple', 'orange', 'banana']
```

- **`join()`**: Joins a list of strings into a single string, with a separator.

Example:

```
 words = ['apple', 'orange', 'banana']
result = ", ".join(words)
print(result)  # Output: apple, orange, banana
```

19. What are iterators, iterable & generators in Python?

- **Iterator**: An iterator is an object that implements two methods: `__iter__()` and `__next__()`. It keeps track of its state and allows sequential access to elements in a collection.
- **Iterable**: An iterable is any object that can return an iterator, e.g., lists, tuples, and dictionaries.
- **Generator**: A generator is a special type of iterator defined using a function with `yield` instead of `return`. It produces items lazily, meaning it generates values one at a time.

Example:

```
 # Generator
def count_up_to(max):
    count = 1
    while count <= max:
        yield count
        count += 1

gen = count_up_to(3)
for num in gen:
    print(num)  # Output: 1, 2, 3
```

20. What is the difference between `xrange` and `range` in Python?

- In Python 2, `range()` generates a list, while `xrange()` generates an iterator, meaning `xrange()` is more memory efficient for large ranges.
- In Python 3, `range()` behaves like `xrange()` in Python 2 and returns an iterator, so `xrange()` no longer exists.

21. Pillars of OOPs.

The four main pillars of **Object-Oriented Programming (OOP)** are:

1. **Encapsulation**: Bundling data and methods into a single unit or class. It restricts direct access to some of the object's attributes and methods, allowing for controlled access.
2. **Abstraction**: Hiding the complex implementation details and showing only the necessary functionality.
3. **Inheritance**: Allowing a new class to inherit attributes and methods from an existing class, promoting code reuse.
4. **Polymorphism**: The ability to use a single interface for different data types, or the ability of a method to behave differently based on the object it is acting upon.

22. How will you check if a class is a child of another class?

You can use the `issubclass()` function to check if a class is a subclass (child) of another class.

Example:
```
 class Parent:
    pass

    •

class Child(Parent):
     pass

 print(issubclass(Child, Parent))  # Output: True
```

23. What is method overriding in Python?

Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.

Example:
```
 class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

dog = Dog()
dog.speak()  # Output: Dog barks
```

-

24. What is the difference between `deepcopy` and `shallow copy` in Python?

- **Shallow Copy**: Copies the reference addresses of the objects, meaning changes in nested objects are reflected in both the original and the copy.
- **Deep Copy**: Copies both the object and all nested objects, so changes in the nested objects do not affect the original.

Example:

```
import copy
original = [[1, 2], [3, 4]]
shallow = copy.copy(original)
deep = copy.deepcopy(original)
```

25. What is Polymorphism?

Polymorphism is a core concept of Object-Oriented Programming (OOP) that allows objects of different classes to be treated as objects of a common base class. The term comes from Greek, meaning "many forms". It allows a single interface to represent different underlying forms (data types). In Python, polymorphism is achieved through method overriding and method overloading.

Polymorphism can be categorized into:

1. **Compile-time Polymorphism (Method Overloading)** - This is not directly supported in Python, but we can achieve it through default arguments.
2. **Runtime Polymorphism (Method Overriding)** - This is supported in Python and occurs when a method in a child class overrides a method in the parent class.

Types of Polymorphism in Python:

1. **Method Overriding** (Runtime Polymorphism): When a subclass provides a specific implementation for a method that is already defined in its parent class.
2. **Operator Overloading**: When you redefine the behavior of operators like `+`, `-`, `*` for custom objects.

Example of Polymorphism in Python:

Method Overriding (Runtime Polymorphism):

```
class Animal:

    def sound(self):
```

```
        print("Animal makes a sound")


class Dog(Animal):
    def sound(self):  # Overriding the sound method of Animal
        print("Dog barks")


class Cat(Animal):
    def sound(self):  # Overriding the sound method of Animal
        print("Cat meows")


def animal_sound(animal):
    animal.sound()  # This will call the appropriate method based on the object type


# Creating objects of Dog and Cat
dog = Dog()
cat = Cat()


# Polymorphism in action
animal_sound(dog)  # Output: Dog barks
animal_sound(cat)  # Output: Cat meows
```

Explanation:

- The `animal_sound` function accepts an `Animal` object and calls its `sound()` method.
- Even though the method is defined in the parent class (`Animal`), the appropriate method is called depending on whether the object is an instance of `Dog` or `Cat`.
- This is an example of **runtime polymorphism** (method overriding) in Python.

Operator Overloading:

Python allows you to define how operators behave with objects. This is also a form of polymorphism.

Example:

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Overloading the + operator to add two vectors
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"

# Creating two Vector objects
v1 = Vector(2, 3)
v2 = Vector(1, 1)

# Using the overloaded + operator
result = v1 + v2
print(result)  # Output: Vector(3, 4)
```

Explanation:

- The `__add__` method is overloaded to define how two `Vector` objects should be added using the `+` operator.
- This is an example of **operator overloading**, a form of polymorphism.

Advantages of Polymorphism:

1. **Code Reusability**: You can write code that works with any subclass of a parent class without knowing the specific subclass.
2. **Flexibility**: Polymorphism allows you to use a common interface for different data types, making code more flexible and easier to maintain.
3. **Extensibility**: It allows you to extend the system by adding new classes without modifying the existing code.

Key Points:

- Polymorphism allows methods to behave differently depending on the object calling them.
- It provides flexibility to write more generic code that works for different types of objects.

Question 1. 2. Which of the following identifier names are invalid and why?
a) Serial_no. b) 1st_Room c) Hundred$ d) Total_Marks e) total-Marks f) Total Marks g) True h) Percentag

Let's evaluate each identifier name in Python to determine if they are valid or invalid:

a) `Serial_no`

- **Valid: This is a valid identifier. It starts with a letter (S), followed by an underscore and other letters/numbers, which is allowed in Python.**

b) `1st_Room`

- **Invalid: This identifier is invalid because Python variable names cannot start with a number. In this case, it starts with `1`, which is not allowed.**

c) `Hundred$`

- **Valid: This is a valid identifier. Python allows the use of the dollar sign (`$`) in identifiers, though it is not commonly used. The identifier does not start with a number or contain any spaces, making it valid.**

d) `Total_Marks`

- **Valid: This is a valid identifier. It starts with a letter and contains an underscore between two words, which is a common practice for multi-word identifiers in Python.**

e) `total-Marks`

- **Invalid: This identifier is invalid because the hyphen (`-`) is not allowed in Python variable names. The hyphen is interpreted as a subtraction operator, so it's not permitted in identifiers.**

f) `Total Marks`

- **Invalid: This identifier is invalid because it contains a space. Python does not allow spaces in variable names, and spaces would cause a syntax error.**

g) `True`

- **Invalid: `True` is a reserved keyword in Python and cannot be used as an identifier. Keywords are predefined words in Python that have a special meaning.**

h) `Percentag`

- **Valid: This is a valid identifier. There are no restrictions on the letters used here, and it does not start with a number or contain any spaces or special characters (other than an underscore, which is valid).**

Summary:

- **Invalid Identifiers:**
  - **`1st_Room` (starts with a number)**
  - **`total-Marks` (contains a hyphen)**
  - **`Total Marks` (contains a space)**
  - **`True` (reserved keyword)**
- **Valid Identifiers:**
  - **`Serial_no`**
  - **`Hundred$`**
  - **`Total_Marks`**
  - **`Percentag`**

Question 1.3.

Let's break down the tasks step by step.

Initial List:

name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]

a) Add an element "freedom_fighter" at the 0th index:

To add "freedom_fighter" at the 0th index, we can use the `insert()` method:

```
name.insert(0, "freedom_fighter")
print(name)
```

Output:

```
['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu']
```

This places "freedom_fighter" at the beginning of the list.

b) Find the output of the following, and explain how:

```
name = ["freedomFighter", "Bapuji", "MOhan", "dash", "karam", "chandra", "gandhi"]
length1 = len(name[-len(name)+1:-1:2])
length2 = len(name[-len(name)+1:-1])
print(length1 + length2)
```

Let's break down the operations:

- `name[-len(name)+1:-1:2]`:
  - `len(name)` is 7 (length of the list).
  - `-len(name) + 1` gives `-7 + 1 = -6`. So the slice starts from index `-6`.
  - `-1` refers to the second-to-last element (index `-1`).
  - `2` is the step, so it takes every second element starting from index `-6` up to index `-1`.

This slice will include the elements: `["MOhan", "karam"]`.

So, `length1 = len(["MOhan", "karam"]) = 2`.

- `name[-len(name)+1:-1]`:
  - **This slice starts from index `-6` (same as before), but it goes until the second-to-last element (`-1`), without skipping any elements.**

This slice will include the elements: `["MOhan", "dash", "karam", "chandra", "gandhi"]`.

So, `length2 = len(["MOhan", "dash", "karam", "chandra", "gandhi"]) = 5`.

Now, let's compute the sum:

length1 + length2 = 2 + 5 = 7

Output:

7

c) Add two more elements "NetaJi" and "Bose" at the end of the list:

To add elements at the end of the list, we use the `extend()` method:

```
name.extend(["NetaJi", "Bose"])
print(name)
```

Output:

['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu', 'NetaJi', 'Bose']

This adds `"NetaJi"` and `"Bose"` to the end of the list.

d) What will be the value of `temp`?

The task is to swap the first and last elements of the list.

```
name = ["Bapuji", "dash", "karam", "chandra", "gandi", "Mohan"]
temp = name[-1]  # temp = 'Mohan'
name[-1] = name[0]  # last element becomes first element: 'Mohan' -> 'Bapuji'
name[0] = temp  # first element becomes 'Mohan'
print(name)
```

Explanation:

- **Initially, `name[-1]` is `"Mohan"`, so `temp = "Mohan"`.**
- **After the swap, the last element becomes the first, and the first element becomes the last.**

Output:

['Mohan', 'dash', 'karam', 'chandra', 'gandi', 'Bapuji']

Summary of Outputs:

- **a) `['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu']`**

- b) `7`
- c) `['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu', 'NetaJi', 'Bose']`
- d) `['Mohan', 'dash', 'karam', 'chandra', 'gandi', 'Bapuji']`

Question 1.4. Find the output of the following. animal= ['Human','cat,'mat,cat,'rat, Human', 'Lion'] print(animal.count('Human')) print(animal.index('rat')(print(len(animal))

Explanation

1. `animal.count('Human')`:
    - The `count()` method counts the occurrences of `'Human'` in the list.
    - `'Human'` appears 2 times in the list, so the output will be `2`.
2. `animal.index('rat')`:
    - The `index()` method returns the index of the first occurrence of `'rat'` in the list.
    - `'rat'` appears at index 4, so the output will be `4`.
3. `len(animal)`:
    - The `len()` function returns the number of elements in the list.
    - The list `animal` contains 7 elements, so the output will be `7`.

Correct Final Output:

2
4
7

Let's break down the question step by step and correct any issues:

The given tuple is:

tuple1 = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "Navneet"}])

a) `print(len(tuple1))`

- Explanation: The `len()` function returns the number of elements in the tuple.
    - `tuple1` contains 8 elements (integers, strings, lists, tuples, and dictionaries).

Output:
print(len(tuple1)) # Output: 8

-

b) `print(tuple1[-1][-1]["name"])`

- **Explanation:**
  - `tuple1[-1]` **refers to the last element of the tuple, which is a list of dictionaries:** `[{"roll_no": 1}, {"name": "Navneet"}]`.
  - `tuple1[-1][-1]` **refers to the last dictionary:** `{"name": "Navneet"}`.
  - `tuple1[-1][-1]["name"]` **fetches the value corresponding to the key** `"name"`**, which is** `"Navneet"`.

Output:
print(tuple1[-1][-1]["name"]) # Output: Navneet

-

c) Fetch the value of `roll_no` from this tuple.

- **Explanation:**
  - `tuple1[-1]` **gives us the list:** `[{"roll_no": 1}, {"name": "Navneet"}]`.
  - `tuple1[-1][0]` **gives us the dictionary:** `{"roll_no": 1}`.
  - `tuple1[-1][0]["roll_no"]` **fetches the value of the key** `"roll_no"`**, which is** `1`.

Output:
print(tuple1[-1][0]["roll_no"]) # Output: 1

-

d) `print(tuple1[-3][1])`

- **Explanation:**
  - `tuple1[-3]` **refers to the third last element of the tuple, which is the tuple** `("sita", "geeta", 22)`.
  - `tuple1[-3][1]` **accesses the second element (index** `1`**) in the tuple, which is** `"geeta"`.

Output:
print(tuple1[-3][1]) # Output: geeta

-

e) Fetch the element 22 from this tuple.

- **Explanation:**
  - `tuple1[-3][2]` **refers to the element at index** `2` **of the tuple** `("sita", "geeta", 22)`**, which is** `22`**.**

Output:
print(tuple1[-3][2]) # Output: 22

-

Final Code:

```
tuple1 = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll_no": 1}, {"name": "Navneet"}])


# a) Print the length of the tuple

print(len(tuple1))  # Output: 8


# b) Print the name from the last dictionary

print(tuple1[-1][-1]["name"])  # Output: Navneet


# c) Fetch the value of 'roll_no'

print(tuple1[-1][0]["roll_no"])  # Output: 1


# d) Print the second element of the tuple at index -3

print(tuple1[-3][1])  # Output: geeta


# e) Fetch the element '22' from the tuple

print(tuple1[-3][2])  # Output: 22
```

Final Output:

8

Navneet

1

geeta

22

1.6. Write a program to display the appropriate message as per the color of signal (RED-Stop/Yellow-Stay/ Green-Go) at the road crossing.

```
def signal_message(color):
    if color.lower() == 'red':
        return "Stop"
    elif color.lower() == 'yellow':
        return "Stay"
    elif color.lower() == 'green':
        return "Go"
    else:
        return "Invalid signal color"

# Get user input for the signal color
signal_color = input("Enter the signal color (Red/Yellow/Green): ")

# Display the appropriate message
print(signal_message(signal_color))
```

Explanation:

1. The `signal_message()` function takes the color of the signal as input and checks it using `if`, `elif`, and `else` statements.
2. The program asks the user to enter the color of the traffic signal.
3. Based on the input color, it returns the appropriate message:
    - `"Stop"` for red
    - `"Stay"` for yellow
    - `"Go"` for green
4. If the color entered is not recognized, it returns `"Invalid signal color"`.

Example Output:

Enter the signal color (Red/Yellow/Green): Green

Go

1.7. Write a program to create a simple calculator performing only four basic operations(+,-,/,*).

```
# Simple calculator program
def calculator():
    op = input("Enter operation (+, -, *, /): ")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    if op == '+':
        return num1 + num2
    elif op == '-':
        return num1 - num2
    elif op == '*':
        return num1 * num2
    elif op == '/':
        return num1 / num2 if num2 != 0 else "Error! Division by zero."
    else:
        return "Invalid operation!"

print(calculator())
```

Explanation:

1. The program takes input for the operation and numbers.
2. It performs the corresponding operation and returns the result.
3. Division by zero is handled with an error message.

Example Output:

```
Enter operation (+, -, *, /): +
Enter first number: 5
Enter second number: 3
8.0
```

1.8. Write a program to find the larger of the three pre-specified numbers using ternary operators.

```
# Program to find the larger of three numbers using ternary operator
a, b, c = 10, 20, 30
largest = a if (a > b and a > c) else (b if b > c else c)
print("The largest number is:", largest)
```

Explanation:

The program uses the ternary operator to compare three numbers `a`, `b`, and `c`. It checks if `a` is greater than both `b` and `c`. If true, `a` is returned as the largest. Otherwise, it checks whether `b` is greater than `c` and returns `b` or `c`.

1.9. Write a program to find the factors of a whole number using a while loop.

```
# Program to find the factors of a number using a while loop
num = int(input("Enter a number: "))
i = 1
while i <= num:
    if num % i == 0:
        print(i, end=" ")
    i += 1
```

Explanation:

The program uses a `while` loop to iterate from `1` to the number itself (`num`). It checks if the number is divisible by `i` using the modulus operator (`%`). If true, it prints `i` as a factor of `num`.

1.10. Write a program to find the sum of all the positive numbers entered by the user. As soon as the user enters a negative number, stop taking in any further input from the user and display the sum.

```
# Program to sum positive numbers and stop on negative input
total_sum = 0
```

```
while True:
    num = float(input("Enter a number: "))
    if num < 0:
        break
    total_sum += num

print("The sum of all positive numbers is:", total_sum)
```

Explanation:
The program continuously takes input from the user. If the user enters a negative number, the loop breaks, and the sum of all positive numbers entered is displayed. It uses a `while` loop and checks if the entered number is negative to stop the input.

1.11. Write a program to find prime numbers between 2 to 100 using nested for loops.

```
# Program to find prime numbers between 2 and 100 using nested for loops
for num in range(2, 101):
    is_prime = True
    for i in range(2, num):
        if num % i == 0:
            is_prime = False
            break
    if is_prime:
        print(num, end=" ")
```

Explanation:
The program checks for prime numbers between `2` and `100`. For each number in the range, it checks if it is divisible by any number from `2` to `num-1`. If a divisor is found, the number is not prime, and it breaks out of the inner loop. If no divisor is found, the number is prime and is printed.

Here is the solution to the problem you described in a question-answer format:

1.12. Write the programs for the following:

Problem Statement:

1. Accept the marks of the student in five major subjects and display the same.
2. Calculate the sum of the marks of all subjects.
3. Divide the total marks by the number of subjects (i.e. 5), calculate percentage = total marks/5 and display the percentage.
4. Find the grade of the student as per the following criteria using `match` & `case`.

Criteria	Grade
percentage > 85	A
percentage < 85 && percentage >= 75	B
percentage < 75 && percentage >= 50	C
percentage > 30 && percentage <= 50	D
percentage < 30	Reappear

Solution:

```
# Program to calculate marks, percentage and grade using match-case


# Accept marks in 5 subjects

marks = []

for i in range(5):

    mark = float(input(f"Enter marks for subject {i+1}: "))

    marks.append(mark)


# Display marks

print("Marks obtained in the 5 subjects:", marks)
```

```
# Calculate total marks and percentage
total_marks = sum(marks)
percentage = total_marks / 5
print("Total Marks:", total_marks)
print("Percentage:", percentage)

# Determine grade using match-case
match percentage:
    case p if p > 85:
        grade = 'A'
    case p if 75 <= p <= 85:
        grade = 'B'
    case p if 50 <= p < 75:
        grade = 'C'
    case p if 30 <= p < 50:
        grade = 'D'
    case _:
        grade = 'Reappear'

# Display the grade
print("Grade:", grade)
```

Explanation:

1. **Accepting Marks: The program accepts marks of the student in 5 subjects using a loop and stores them in a list.**
2. **Displaying Marks: It then displays the marks entered by the user.**

3. **Calculating Total and Percentage: The program calculates the total marks by summing the list and calculates the percentage by dividing the total marks by the number of subjects (5).**
4. **Determining Grade: Using Python's `match` and `case` constructs, the program determines the grade based on the percentage. The grade is assigned based on the criteria mentioned.**
5. **Output: Finally, it displays the calculated percentage and grade.**

Example Output:

Enter marks for subject 1: 90

Enter marks for subject 2: 80

Enter marks for subject 3: 85

Enter marks for subject 4: 70

Enter marks for subject 5: 60

Marks obtained in the 5 subjects: [90.0, 80.0, 85.0, 70.0, 60.0]

Total Marks: 385.0

Percentage: 77.0

Grade: B

1.14. Gravitational Interactions between Earth, Moon, and Sun

Formula:

F=G⋅m1⋅m2r2F = \frac{G \cdot m_1 \cdot m_2}{r^2}

Where:

- **FF is the gravitational force.**
- **GG is the gravitational constant (6.674×10−116.674 \times 10^{-11} m^3 kg^{-1} s^{-2}).**
- **m1,m2m_1, m_2 are the masses of two objects.**
- **rr is the distance between the objects.**

Given:

- **Mass of Earth: 5.972×1024 kg5.972 \times 10^{24} \, \text{kg}**
- **Mass of Moon: 7.34767309×1022 kg7.34767309 \times 10^{22} \, \text{kg}**
- **Mass of Sun: 1.989×1030 kg1.989 \times 10^{30} \, \text{kg}**
- **Distance Earth-Sun: 1.496×1011 m1.496 \times 10^{11} \, \text{m}**
- **Distance Moon-Earth: 3.844×108 m3.844 \times 10^{8} \, \text{m}**

Calculation:

- **Gravitational force between Earth and Sun:**

Fearth-sun=6.674×10−11×5.972×1024×1.989×1030(1.496×1011)2F_{\text{earth-sun}} = \frac{6.674 \times 10^{-11} \times 5.972 \times 10^{24} \times 1.989 \times 10^{30}}{(1.496 \times 10^{11})^2}

- **Gravitational force between Moon and Earth:**

Fmoon-earth=6.674×10−11×7.34767309×1022×5.972×1024(3.844×108)2F_{\text{moon-earth}} = \frac{6.674 \times 10^{-11} \times 7.34767309 \times 10^{22} \times 5.972 \times 10^{24}}{(3.844 \times 10^8)^2}

Conclusion:

The gravitational force between the Earth and the Sun is much stronger than between the Moon and Earth, due to the Sun's mass being much greater than the Moon's. However, the Moon's proximity to Earth results in a stronger attraction between the Moon and Earth than Earth's attraction to the Sun.

(Q.2 - Q10) - OOPS

Ans-

https://github.com/patiluday3101/Final_assinment_pw.git

11. Define a Python module named `constants.py` containing constants like pi and the speed of light.

Answer:

A Python module `constants.py` can be created to store constant values used across programs.

Example:

```
# constants.py

PI = 3.14159

SPEED_OF_LIGHT = 3e8  # in meters/second
```

---

12. Write a Python module named `calculator.py` containing functions for addition, subtraction, multiplication, and division.

Answer:
The module `calculator.py` can be created for basic arithmetic operations.

Example:

```
# calculator.py

def add(a, b):
    return a + b


def subtract(a, b):
    return a - b


def multiply(a, b):
    return a * b


def divide(a, b):
    return a / b if b != 0 else "Division by zero error"
```

---

13. Implement a Python package structure for a project named `ecommerce`, containing modules for product management and order processing.

Answer:
Package structure:

```
ecommerce/
    __init__.py
    product_management.py
    order_processing.py
```

Example contents:

```
# product_management.py

def add_product(name, price):

    print(f"Product {name} added with price {price}")


# order_processing.py

def process_order(order_id):

    print(f"Processing order {order_id}")
```

---

14. Implement a Python module named `string_utils.py` containing functions for string manipulation, such as reversing and capitalizing strings.

Answer:
Example:

```
# string_utils.py

def reverse_string(s):

    return s[::-1]


def capitalize_string(s):

    return s.capitalize()
```

---

15. Write a Python module named `file_operations.py` with functions for reading, writing, and appending data to a file.

Answer:
Example:

```
# file_operations.py

def write_to_file(filename, data):
    with open(filename, 'w') as f:
        f.write(data)


def read_from_file(filename):
    with open(filename, 'r') as f:
        return f.read()


def append_to_file(filename, data):
    with open(filename, 'a') as f:
        f.write(data)
```

---

16. Write a Python program to create a text file named "employees.txt" and write the details of employees, including their name, age, and salary, into the file.

Answer:
Example:

```
with open("employees.txt", 'w') as file:
    employees = [("John", 30, 50000), ("Jane", 25, 60000)]
    for name, age, salary in employees:
        file.write(f"Name: {name}, Age: {age}, Salary: {salary}\n")
```

---

17. Develop a Python script that opens an existing text file named "inventory.txt" in read mode and displays the contents of the file line by line.

Answer:
Example:

```
with open("inventory.txt", 'r') as file:

    for line in file:

        print(line.strip())
```

---

18. Create a Python script that reads a text file named "expenses.txt" and calculates the total amount spent on various expenses listed in the file.

Answer:
Example:

```
total = 0

with open("expenses.txt", 'r') as file:

    for line in file:

        total += float(line.strip())

print(f"Total expenses: {total}")
```

---

19. Create a Python program that reads a text file named "paragraph.txt" and counts the occurrences of each word in the paragraph, displaying the results in alphabetical order.

Answer:
Example:

```
from collections import Counter


with open("paragraph.txt", 'r') as file:

    words = file.read().lower().split()

word_counts = Counter(words)
```

```
for word, count in sorted(word_counts.items()):
    print(f"{word}: {count}")
```

---

20. What do you mean by Measure of Central Tendency and Measures of Dispersion? How can it be calculated?

Answer:

- **Measure of Central Tendency: Represents the center or typical value of a dataset (e.g., Mean, Median, Mode).**
- **Measures of Dispersion: Represents the spread of data around the central value (e.g., Range, Variance, Standard Deviation).**

---

21. What do you mean by skewness? Explain its types. Use a graph to show.

Answer:

- **Skewness: Indicates the asymmetry of a dataset's distribution.**
  - **Positive Skew: Tail on the right (e.g., income data).**
  - **Negative Skew: Tail on the left (e.g., age of death data).**
  - **Symmetric: Equal tails (e.g., normal distribution).**

---

22. Explain Probability Mass Function (PMF) and Probability Density Function (PDF). What is the difference?

Answer:

- **PMF: Defines probabilities for discrete random variables (e.g., dice roll outcomes).**
- **PDF: Represents probabilities for continuous random variables (e.g., height distribution).**
- **Difference: PMF sums up to 1, while PDF integrates to 1 over the range.**

---

23. What is correlation? Explain its types in detail. What are the methods of determining correlation?

Answer:

- **Correlation: Measures the relationship between two variables.**
  - **Types: Positive, Negative, No correlation.**
  - **Methods: Pearson (linear), Spearman (rank-based), Kendall (ordinal).**

25. Discuss the 4 differences between correlation and regression.

Answer:

Aspect	Correlation	Regression
Purpose	Measures the strength and direction of a relationship between variables.	Predicts one variable based on another.
Variables	Both variables are treated symmetrically.	One is independent (predictor), and the other is dependent.
Values	Ranges between -1 and +1.	Provides an equation for prediction.
Dependency	Does not imply causation.	Implies dependency of one variable on another.

---

26. Find the most likely price at Delhi corresponding to the price of Rs. 70 at Agra from the following data.

Given: Coefficient of correlation $r=+0.8$,
Standard deviation of prices: $\sigma_{\text{Agra}} = 3$, $\sigma_{\text{Delhi}} = 6$,
Mean prices: $\mu_{\text{Agra}} = 68$, $\mu_{\text{Delhi}} = 75$.

Solution:
Using regression equation:
$$y - \mu_y = r \times \frac{\sigma_y}{\sigma_x} \times (x - \mu_x)$$
Substitute values:
$$y - 75 = 0.8 \times \frac{6}{3} \times (70 - 68)$$
$$y - 75 = 0.8 \times 2 \times 2$$

y=75+3.2y = 75 + 3.2
Answer: Most likely price at Delhi = 78.2 Rs.

---

27. Partially destroyed correlation data problem.

Given:
Regression equations: 8x−10y=−668x - 10y = -66 and 40x−18y=21440x - 18y = 214.
Variance of x=9x = 9.

Solution:
(a) Find the mean values μx\mu_x and μy\mu_y:
Solve the regression equations for xx and yy simultaneously.

(b) Coefficient of correlation rr:
r2=bxy⋅byxr^2 = b_{xy} \cdot b_{yx}

(c) Standard deviation of yy:
σy=σx∣bxy∣\sigma_y = \frac{\sigma_x}{|b_{xy}|}

---

28. What is Normal Distribution? What are the four assumptions of Normal Distribution?

Answer:

- **Normal Distribution: A bell-shaped curve describing data distribution where the mean, median, and mode are equal.**

Four Assumptions:

1. **Data is continuous and random.**
2. **Symmetric about the mean.**
3. **Follows empirical rule: μ±σ\mu \pm \sigma, μ±2σ\mu \pm 2\sigma, μ±3σ\mu \pm 3\sigma.**
4. **No outliers.**

---

29. Characteristics or Properties of the Normal Distribution Curve.

1. **Bell-shaped and symmetric.**
2. **Mean = Median = Mode.**
3. **Total area under the curve is 1.**
4. **68.26%, 95.44%, and 99.73% fall within 1, 2, and 3 standard deviations, respectively.**

---

30. Correct options about Normal Distribution Curve.

Correct options:
(b), (c), (d), and (e).

---

31. Normal distribution problem.

Given: Mean = 60, Standard Deviation = 10.

Solution:
Use Z-scores to calculate percentages for:
(i) 6060 to 7272.
(ii) 5050 to 6060.
(iii) Beyond 7272.
(iv) 7070 to 8080.

---

32. Proportion of students scoring marks.

Given: Mean = 49, σ=6\sigma = 6, Total = 15000 students.

Solution:
Use Z-scores for marks >55>55 and >70>70, and calculate proportions using the normal distribution table.

---

33. Height of 500 students.

Given: Mean = 65, σ=5\sigma = 5, Total = 500 students.

Solution:
Calculate proportions for >70>70 and 6060 to 7070 using Z-scores and multiply by 500.

---

34. What is the statistical hypothesis? Errors in hypothesis testing.

Answer:

- **Statistical Hypothesis: A claim about a population parameter tested using sample data.**
- **Errors in Hypothesis Testing:**

- **Type I: Rejecting a true null hypothesis.**
- **Type II: Failing to reject a false null hypothesis.**

Samples:

- **Large Samples: n>30n > 30.**
- **Small Samples: n≤30n \leq 30.**

---

35. Hypothesis test using chi-square distribution.

Given: Sample size = 25, s=9.0s = 9.0, H0:σ=10.5H_0: \sigma = 10.5.

Solution:
Use chi-square test statistic:
χ2=(n−1)⋅s2σ2\chi^2 = \frac{(n-1) \cdot s^2}{\sigma^2}

---

37. Chi-square test for uniform distribution of grades.

Solution:

1. **Null hypothesis: Grades are uniformly distributed.**
2. **Expected frequency: Total FrequencyNumber of Grades\frac{\text{Total Frequency}}{\text{Number of Grades}}.**
3. **Use chi-square formula:**
   χ2=∑(Oi−Ei)2Ei\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}
   Compare χ2\chi^2 with critical value.

38. ANOVA Test (Analysis of Variance):

Given Data:

Water Temp	Detergent A	Detergent B	Detergent C
Cold Water	57	55	67
Warm Water	49	52	68

Hot Water 54 46 58

Steps to Perform ANOVA:

1. **State the null hypothesis H0H_0: The means of all groups are equal.**
2. **Calculate the total sum of squares (SST), sum of squares between groups (SSB), and sum of squares within groups (SSW).**
3. **Compute F-statistic:**
   F=SSB/df betweenSSW/df withinF = \frac{\text{SSB/df between}}{\text{SSW/df within}}
4. **Compare F-value with critical value to decide H0H_0.**

---

39. Create a basic Flask route for "Hello, World!":

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def home():

    return "Hello, World!"


if __name__ == '__main__':

    app.run(debug=True)
```

---

40. Setting up a Flask application to handle POST form submissions:

1. **Create a form in HTML:**

```
<form method="POST" action="/submit">

  <input type="text" name="name">

  <button type="submit">Submit</button>
```

```
</form>
```

2. **Flask route to handle the form:**

```
from flask import Flask, request

app = Flask(__name__)


@app.route('/submit', methods=['POST'])

def submit():

    name = request.form['name']

    return f"Hello, {name}"


if __name__ == '__main__':

    app.run(debug=True)
```

---

41. Flask route accepting a parameter in the URL:

```
@app.route('/user/<name>')

def user(name):

    return f"Welcome, {name}!"
```

---

42. Implementing user authentication in Flask:

1. **Use Flask-Login or Flask-Security.**
2. **Steps:**
    - **Create a user model.**
    - **Hash passwords with libraries like `bcrypt`.**
    - **Authenticate using session management.**

---

43. Connect Flask app to SQLite using SQLAlchemy:

```
from flask_sqlalchemy import SQLAlchemy


app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)
```

---

44. Create a RESTful API endpoint in Flask that returns JSON data:

```
from flask import jsonify


@app.route('/api/data')
def api_data():
    data = {"message": "Hello, World!"}
    return jsonify(data)
```

---

45. Using Flask-WTF for form creation and validation:

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired
```

```
class MyForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    submit = SubmitField('Submit')
```

---

46. Implement file uploads in Flask:

```
@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    file.save(f"/uploads/{file.filename}")
    return "File uploaded successfully!"
```

---

47. Steps to create a Flask blueprint:

Define blueprint:

```
from flask import Blueprint
bp = Blueprint('my_blueprint', __name__)
```

1.

Register blueprint in app:

```
app.register_blueprint(bp)
```

2.

---

48. Deploy Flask app using Gunicorn and Nginx:

1. Install Gunicorn and Nginx.

Use Gunicorn to serve the app:

```
gunicorn -w 4 -b 0.0.0.0:8000 app:app
```

2. Configure Nginx to proxy requests to Gunicorn.

---

49. Fully functional Flask web application:

Features: Signup, Sign-in, MongoDB integration, and greeting page.

Steps:

1. Use `flask-pymongo` for MongoDB connection.
2. Create forms for signup and sign-in.
3. Store user data in MongoDB.
4. Redirect to "Hello Geeks" page after login.

50.Machine Learning:

Machine Learning Questions and Answers

---

1. Difference Between Series & DataFrames

- **Series: A one-dimensional labeled array, similar to a list or column in a table.**
- **DataFrame: A two-dimensional, tabular data structure with labeled rows and columns.**

---

2. Create `Travel_Planner` Database and Read Table with Pandas

- **SQL:**

```
CREATE DATABASE Travel_Planner;
USE Travel_Planner;
CREATE TABLE bookings (
    user_id INT,
    flight_id INT,
    hotel_id INT,
    activity_id INT,
    booking_date DATE
);
INSERT INTO bookings VALUES (1, 101, 201, 301, '2024-12-01'), (2, 102, 202, 302, '2024-12-02');
```

- **Python Code:**

```
import pandas as pd
import mysql.connector

connection = mysql.connector.connect(
    host="localhost",
    user="your_user",
    password="your_password",
    database="Travel_Planner"
)
query = "SELECT * FROM bookings"
df = pd.read_sql(query, connection)
print(df)
```

---

3. Difference Between `loc` and `iloc`

- **`loc`: Label-based indexing to access data by labels or names.**
- **`iloc`: Position-based indexing to access data by numerical indices.**

---

4. Difference Between Supervised and Unsupervised Learning

- **Supervised Learning: Trains on labeled data (input-output pairs). Examples: Regression, Classification.**
- **Unsupervised Learning: Finds patterns in unlabeled data. Examples: Clustering, Dimensionality Reduction.**

---

5. Explain Bias-Variance Tradeoff

- **Bias: Error due to overly simplistic assumptions (underfitting).**
- **Variance: Error due to model sensitivity to training data (overfitting).**
- **Tradeoff: Balance between bias and variance to minimize total error.**

---

6. What Are Precision, Recall, and Accuracy?

- **Precision: True positives out of total predicted positives.**
  Precision=TPTP+FPPrecision = \frac{TP}{TP + FP}
- **Recall: True positives out of actual positives.**
  Recall=TPTP+FNRecall = \frac{TP}{TP + FN}
- **Accuracy: Correct predictions out of total predictions.**
  Accuracy=TP+TNTotalAccuracy = \frac{TP + TN}{Total}

---

7. What is Overfitting and How to Prevent It?

- **Overfitting: Model performs well on training data but poorly on unseen data.**
- **Prevention:**
    1. **Use cross-validation.**
    2. **Reduce model complexity.**
    3. **Add regularization.**
    4. **Increase training data.**

---

8. Explain Cross-Validation

- **A technique to evaluate model performance by splitting the dataset into multiple training and testing subsets. Example: k-fold cross-validation.**

---

9. Classification vs. Regression

- **Classification: Predicts categorical outcomes (e.g., spam or not spam).**
- **Regression: Predicts continuous outcomes (e.g., house prices).**

---

10. What is Ensemble Learning?

- **Combines predictions from multiple models to improve accuracy. Examples: Bagging, Boosting.**

---

11. What is Gradient Descent?

- **Optimization algorithm to minimize the cost function by iteratively updating parameters in the direction of the steepest descent.**

---

12. Batch vs. Stochastic Gradient Descent

- **Batch Gradient Descent: Uses the entire dataset for parameter updates.**
- **Stochastic Gradient Descent (SGD): Uses one data point per iteration.**

---

13. What is the Curse of Dimensionality?

- **As the number of features increases, the data becomes sparse, reducing the effectiveness of distance-based metrics.**

---

14. L1 vs. L2 Regularization

- **L1 Regularization (Lasso): Penalizes absolute values of coefficients; promotes sparsity.**
- **L2 Regularization (Ridge): Penalizes squared values of coefficients; prevents large weights.**

---

15. What is a Confusion Matrix?

- **A table summarizing model performance with counts of:**
  - **True Positives (TP)**
  - **True Negatives (TN)**
  - **False Positives (FP)**
  - **False Negatives (FN).**

---

16. Define AUC-ROC Curve

- **AUC (Area Under Curve): Measures the ability of a classifier to distinguish between classes.**
- **ROC (Receiver Operating Characteristic) Curve: Plots True Positive Rate (TPR) vs. False Positive Rate (FPR).**

---

17. Explain the k-Nearest Neighbors Algorithm

- **A supervised learning algorithm that classifies data based on the majority class of its nearest neighbors in the feature space.**

---

18. What is a Support Vector Machine (SVM)?

- **A supervised learning algorithm that finds the optimal hyperplane to separate data into different classes.**

---

19. How Does the Kernel Trick Work in SVM?

- **Maps data into a higher-dimensional space to make it linearly separable using kernel functions like:**
  - **Linear**
  - **Polynomial**
  - **Radial Basis Function (RBF).**

---

20. Types of Kernels in SVM

- **Linear Kernel: For linearly separable data.**
- **Polynomial Kernel: For data with curved boundaries.**
- **RBF Kernel: Handles complex, non-linear relationships.**

---

21. What is the Hyperplane in SVM?

- **A decision boundary separating classes, determined by maximizing the margin between data points of different classes.**

---

22. Pros and Cons of SVM

- **Pros: Effective for small datasets, works well with high-dimensional data.**
- **Cons: Computationally expensive for large datasets, sensitive to parameter tuning.**

---

23. Hard Margin vs. Soft Margin SVM

- **Hard Margin: No misclassification allowed; requires perfectly separable data.**
- **Soft Margin: Allows some misclassification to handle overlapping data.**

---

24. Describe the Process of Constructing a Decision Tree

1. **Calculate the impurity (e.g., Gini, entropy) for each split.**
2. **Choose the split that minimizes impurity.**
3. **Recursively split until stopping criteria are met.**

---

25. What is Information Gain in Decision Trees?

- **A measure of the reduction in entropy after a split. Higher information gain indicates a better split.**

---

26. Explain Gini Impurity

- **Measures the probability of incorrect classification. Lower Gini implies better splits.**

---

27. Advantages and Disadvantages of Decision Trees

- **Advantages: Easy to interpret, handles both numerical and categorical data.**
- **Disadvantages: Prone to overfitting, sensitive to small data changes.**

---

28. How Do Random Forests Improve Upon Decision Trees?

- **Combines multiple decision trees to reduce overfitting and improve generalization.**

---

29. How Does a Random Forest Work?

1. **Creates multiple decision trees using bootstrap samples.**
2. **Aggregates predictions (e.g., majority vote for classification).**

---

30. What is Bootstrapping in Random Forests?

- **A resampling method to create diverse training datasets by sampling with replacement.**

---

31. Explain Feature Importance in Random Forests

- **Measures the contribution of each feature to the model's predictive power, typically using metrics like Gini importance.**

---

32. Key Hyperparameters of Random Forests

- **n_estimators: Number of trees.**
- **max_depth: Maximum depth of trees.**
- **min_samples_split: Minimum samples to split a node.**
- **Effect: Controls model complexity and prevents overfitting.**

---

33. Describe the Logistic Regression Model

- **A statistical model for binary classification that uses the sigmoid function to predict probabilities.**

---

34. What is the Sigmoid Function?

- **A mathematical function that maps input values to a range between 0 and 1. σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}**

---

35. Explain Cost Function in Logistic Regression

- **Measures the error between predicted probabilities and actual labels, optimized using gradient descent.**

---

36. Logistic Regression for Multiclass Classification

- **Uses techniques like One-vs-Rest (OvR) or Softmax Regression.**

---

37. What is XGBoost?

- **An efficient and scalable gradient boosting library for supervised learning tasks.**

---

38. How Does XGBoost Handle Missing Values?

- **Automatically learns the best direction (left or right split) for missing data.**

---

39. Key Hyperparameters in XGBoost

- **learning_rate: Shrinks contribution of each tree.**
- **n_estimators: Number of boosting rounds.**
- **max_depth: Depth of trees.**
- **Effect: Balances bias-variance tradeoff.**

---

40. Advantages and Disadvantages of XGBoost

- **Advantages: High accuracy, handles missing data, parallel processing.**
- **Disadvantages: Computationally expensive, prone to overfitting without tuning.**

Machine learning Practical question:

1. Take any project from PW Experience Portal form machine learning domain.And make an end to end project with all the necessary documents.
Link: https://experience.pwskills.com/

2)Do the EDA on the given dataset: Lung cancer, and extract some useful information from this. Dataset Description: Lung cancer is one of the most prevalent and deadly forms of cancer worldwide, presenting significant challenges in early detection and effective treatment. To aid in the global effort to understand and combat this disease, we are excited to introduce our comprehensive Lung Cancer Dataset. Link: Lung Cancer
Ans-
https://github.com/patiluday3101/Final_assinment_pw.git

3) 3. Do the Eda on this Dataset:Presidential Election Polls 2024 Dataset and extract useful information from this: Link: Dataset: Nationwide Russian election poll data from March 04, 2024 Dataset Description: This dataset comprises the results of a nationwide presidential election poll conducted on March 4, 2024.The data offers various insights but does not align with the official election results. You are encouraged to create your notebooks and delve into the data for further exploration.

Ans-
https://github.com/patiluday3101/Final_assinment_pw.git