**i. Object-oriented programming (OOP)**

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects." Objects can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). OOP aims to model real-world entities and relationships in a programming environment, facilitating code reuse, modularity, and scalability. Key principles of OOP include encapsulation, inheritance, and polymorphism.

**ii. Encapsulation**

Encapsulation is an OOP principle that involves bundling the data (attributes) and methods (functions) that operate on the data into a single unit or class. It also restricts direct access to some of an object's components, which is a means of preventing unintended interference and misuse. By exposing only necessary parts of an object through public methods, encapsulation helps to protect the object's internal state and maintain control over how it is accessed and modified.

**iii. Subclass**

A subclass (also known as a derived class or child class) is a class that inherits attributes and methods from another class, known as the parent class or superclass. The subclass can add its own attributes and methods, or it can override the inherited methods to provide specific behaviors. This relationship allows for code reuse and the creation of more specialized classes based on more general ones.

**iv. Parent class**

A parent class (also known as a superclass or base class) is a class that is extended or inherited by one or more subclasses. The parent class defines common attributes and methods that the subclasses can use, and potentially override or extend. This allows for a hierarchical classification of classes, promoting code reuse and logical organization of class structures.

**v. Function**

A function is a block of organized, reusable code that performs a single, specific task. Functions are defined by a name and can take input parameters, process data, and return an output. They help in breaking down complex problems into smaller, manageable parts, improving code readability, maintainability, and reusability. Functions are fundamental constructs in many programming languages, including those that support OOP.

### b) Python array called level.

i. Express level and its elements in an array form.

level = ["new", "geek", "hack", "pro"]

#### ii. Using append, add another level, guru, to the array members.

level.append("guru")

After appending, the level array will be:

["new", "geek", "hack", "pro", "guru"]

#### iii. Write an expression to delete the second element from the array.

del level[1]

After deletion, the level array will be:

["new", "hack", "pro", "guru"]

#### iv. Use a standard Python method to deduce the length of the array, level.

len(level)

This expression will return 4.

#### v. Write an expression to get the second value from the level array.

second\_value = level[1]

After the deletion in part iii, this will be "hack".

#### vi. Briefly explain a dictionary as used in Python programming. Provide an example of a dictionary.

A dictionary in Python is a collection of key-value pairs where each key is unique and is used to store and retrieve data efficiently. Dictionaries are mutable, meaning they can be changed after creation, and they allow for fast access, insertion, and deletion of elements based on keys.

Example of a dictionary:

student = {

"name": "Alice",

"age": 21,

"major": "Computer Science"

}

C.) class Student:

def \_\_init\_\_(self, firstName, lastName, institution, averageScore, age):

self.firstName = firstName

self.lastName = lastName

self.institution = institution

self.averageScore = averageScore

self.age = age

def \_\_str\_\_(self):

return (f"Student Name: {self.firstName} {self.lastName}\n"

f"Institution: {self.institution}\n"

f"Average Score: {self.averageScore}\n"

f"Age: {self.age}")

# Example of creating a Student object

student1 = Student("John", "Doe", "Example University", 85.5, 20)

# Printing the student object

print(student1)

d) Write a Python program to reverse a given string.

*Sample String:* "decimal"

*Expected Output:* "lamiced"

def reverse\_string(s):

return s[::-1]

sample\_string = "decimal"

print(reverse\_string(sample\_string))

# Output: "lamiced"

e) Write a Python function to find the maximum of three temperature records with variable

names, temp1, temp2, and temp3.

def find\_max\_temp (temp1, temp2, temp3):

return max(temp1, temp2, temp3)

temp1 = 29

temp2 = 35

temp3 = 19

print(find\_max\_temp (temp1, temp2, temp3))

f) i. Briefly explain and illustrate **Dictionaries** as used in Python program.

Dictionaries are collections of key-value pairs where each key is unique. They allow for efficient retrieval, addition, and deletion of values based on their keys.

Example:

student\_info = {

'name': 'John Doe',

'age': 20,

'institution': 'ABC University'

}

g) Write a Python function that checks whether a passed string is a palindrome or not. (A

palindrome reads same when spelt backward or forward. Eg. Madam)

def is\_palindrome(s):

return s == s[::-1]

# Example usage

print(is\_palindrome("madam"))

# Output: True

print(is\_palindrome("hello"))

# Output: False