OOPS

Theory/Conceptual based

What is OOPs?

- OOP in Python stands for Object-Oriented Programming.
- It uses objects and classes to organize code in a way that models real-world things and their interactions, making the code more modular and reusable.

Difference between Procedural programming and OOPs?

Feature	Procedural Programming	Object-Oriented Programming (OOP)
Basic Concept	Focuses on functions and procedures	Focuses on objects and classes
Approach	Follows a step-by-step approach	Follows a modular approach using objects
Data Handling	Data and functions are separate	Data and methods are encapsulated in objects
Code Reusability	Limited reuse through functions	High reusability through inheritance and polymorphism
Data Access	Global data can be accessed by any function	Controlled access using encapsulation
Complexity Management	Less suited for large and complex programs	Better suited for large and complex programs

Modularity	Achieved through functions	Achieved through classes and objects
State Management	State is often managed globally or passed between functions	State is managed within objects
Example Languages	C, Fortran, Pascal	Python, Java, C++
Maintenance	Can be harder to maintain for large codebases	Easier to maintain due to encapsulation and modularity
Debugging	Can be more difficult due to less modularity	Easier due to encapsulation and better organization
Real-world Modeling	Less natural for real-world modeling	More natural for real-world modeling

What are the primary characteristics of OOPs?

The primary characteristics of Object-Oriented Programming (OOP) are:

1. **Encapsulation**: Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, typically a class.It restricts direct access to some of the object's components, which can prevent the accidental modification of data. Encapsulation also makes the code more modular and easier to maintain.
2. **Abstraction**: Abstraction involves hiding the complex implementation details and showing only the essential features of the object. It reduces complexity and allows the programmer to focus on interactions at a higher level.
3. **Inheritance**: Inheritance is a mechanism by which one class (subclass or derived class) can inherit attributes and methods from another class

(superclass or base class). It promotes code reuse and establishes a natural hierarchical relationship between classes.
4. **Polymorphism**: Polymorphism allows methods to do different things based on the object it is acting upon, even though they share the same name. It enables a single interface to represent different underlying forms (data types).

What are the advantages and disadvantages of OOPs?

- Object-oriented programming (OOP) offers several advantages, including enhanced modularity, code reusability, and scalability, as it allows for the creation of objects that encapsulate both data and behaviours.
- This encapsulation promotes clear and organised code, making maintenance and debugging more manageable.
- Additionally, OOP facilitates inheritance, enabling new classes to build upon existing ones, and polymorphism, which allows for the use of a single interface to represent different underlying forms (data types).
- However, OOP also has disadvantages, such as potentially increased complexity and learning curve, as designing and understanding object hierarchies can be challenging. Additionally, it can sometimes lead to performance overhead due to abstraction layers and the management of numerous objects.

What is the difference between a class and an object?

Feature	**Class**	**Object**
Definition	A class is a blueprint or template for creating objects. It defines a set of attributes and methods that the objects created from the class will have.	An object is an instance of a class. It is created using the class blueprint and represents a specific implementation with its own state.
Nature	Conceptual; it defines properties and behaviors.	Concrete; it is an actual entity in memory.
Creation	Defined using the class keyword in Python.	Created using the class by instantiating it.
Usage	Used to define structure and behavior common to all objects of its type.	Used to interact with the system and perform

		operations defined by the class.
Example	python class Dog: def __init__(self, name): self.name = name def bark(self): print(f"{self.name} says woof!")	python my_dog = Dog("Buddy") my_dog.bark() # Output: Buddy says woof!
Data and Methods	Contains definitions for attributes (variables) and methods (functions).	Contains actual values for attributes and can call methods defined by its class.
Memory Allocation	No memory is allocated until an object is created from it.	Memory is allocated for attributes when an object is instantiated.

What is the constructor in a Python class?

In Python, a constructor is a special method that is automatically called when an instance (object) of a class is created. The purpose of a constructor is to initialise the object's attributes and perform any setup necessary for the object. In Python, the constructor method is named __init__.

Key Points About the Constructor (__init__):

1. Method Name: The constructor method is always named __init__.
2. Initialization: It is used to initialise the attributes of the class.
3. Automatic Invocation: It is automatically called when a new object of the class is instantiated.

What are various types of constructors?

In Python, constructors can be categorised into several types based on their purpose and usage. The main types of constructors are:

- Default Constructor
- Parameterized Constructor
- Non-Parameterized Constructor

Default Constructor : A default constructor is a constructor that does not accept any arguments except for self. It initialises objects with default values.
Parameterized Constructor : A parameterized constructor is a constructor that accepts one or more parameters. It is used to initialise an object with specific values provided at the time of creation.
Non-Parameterized Constructor : A non-parameterized constructor is similar to the default constructor but is explicitly defined without parameters other than self.

Do we require a parameter for constructors?

In Python, constructors do not necessarily require parameters, but they can have parameters if needed. The constructor in Python is defined using the __init__ method. Whether you include parameters in the constructor depends on your use case and what information you need to initialise an object of the class.

- No Parameter Constructor (Default Constructor)
  A constructor without parameters is used to initialise objects with default values.
- Constructor with Parameters (Parameterized Constructor)
  A constructor with parameters is used to initialise objects with specific values provided at the time of creation.
- Flexible Constructors with Default Parameter Values
  You can also create constructors that have parameters with default values, allowing them to be used with or without arguments.

What is Abstraction?

- Abstraction is one of the fundamental principles of Object-Oriented Programming.
- It involves the process of hiding the complex implementation details and showing only the essential features and behaviours of an object.
- The goal of abstraction is to reduce complexity and allow the programmer to focus on interactions at a higher level.

What is an abstract class?

An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed. It serves as a blueprint for other classes. An abstract class can include abstract methods, which are methods that are declared but contain no implementation. Subclasses of the abstract class must provide implementations for these abstract methods.
Benefits of Using Abstract Classes:

- Enforces Consistency: Ensures that all subclasses implement the necessary methods.
- Promotes Reusability: Common functionality can be defined once in the abstract class and reused by all subclasses.
- Improves Code Organization: Helps in organising code in a more structured and understandable way.

How many instances can be created for an abstract class?

In Python, you cannot create instances directly from an abstract class. Abstract classes are designed to be inherited by subclasses that provide implementations for the abstract methods. An abstract class serves as a blueprint, and its primary purpose is to define a common interface for its subclasses.

What is the use of the @abstractmethod decorator?

The @abstractmethod decorator in Python is used to declare a method as abstract in an abstract base class (ABC). An abstract method is a method that is declared but contains no implementation. Subclasses of the abstract base class are required to provide implementations for all abstract methods defined in the base class. This is a way to enforce that certain methods must be implemented by any subclass of the abstract class.

What is the purpose of the self keyword in methods?

The self keyword in Python is used within class methods to refer to the instance of the class upon which the method is being called. It allows access to the attributes and other methods of the class in a clear and unambiguous way.

What is encapsulation? How do you implement it in Python?

Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP). It refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, typically a class. Encapsulation restricts direct access to some of an object's components, which can prevent the accidental modification of data. This is often achieved through the use of access modifiers that control the visibility of attributes and methods.
Implementing Encapsulation in Python:
In Python, encapsulation can be achieved using access modifiers. Python supports three types of access modifiers:

Public:

- Attributes and methods are accessible from outside the class.
- No special prefix is used.

Protected:

- Attributes and methods are accessible within the class and its subclasses.
- A single underscore (_) prefix is used.

Private:

- Attributes and methods are accessible only within the class.
- A double underscore (__) prefix is used.

What are the access modifiers?

In Python, access modifiers are used to set the visibility and accessibility of class attributes and methods. They help in implementing encapsulation by controlling how data and methods can be accessed from outside the class. Python has three main types of access modifiers:

1. Public
2. Protected
3. Private

Public Access Modifier

- Syntax: No special prefix is used.
- Accessibility: Attributes and methods declared as public are accessible from anywhere—both inside and outside the class.
- Usage: By default, all attributes and methods in a class are public.

Protected Access Modifier

- Syntax: A single underscore _ prefix is used.
- Accessibility: Attributes and methods declared as protected are accessible within the class and its subclasses but should not be accessed directly from outside the class. This is more of a convention than a strict rule in Python.
- Usage: Used to indicate that the attribute or method is intended for internal use or use by subclasses.

Private Access Modifier

- Syntax: A double underscore __ prefix is used.
- Accessibility: Attributes and methods declared as private are accessible only within the class where they are defined. They are not accessible from outside

the class or by subclasses directly. Python uses name mangling to change the name of the attribute so that it is not easily accessible.
- Usage: Used to restrict access to the most sensitive parts of the class.

What is the default access specifier in class definition?

In Python, the default access specifier for class members (attributes and methods) is public. This means that class members can be accessed from anywhere, both inside and outside the class.

What is information hiding in OOP?

Information hiding is a fundamental concept in object-oriented programming (OOP) that refers to the practice of restricting access to the internal state and implementation details of an object. This concept is closely related to encapsulation, but it specifically emphasises the hiding of internal details from outside access and manipulation.

What is the purpose of the pass statement in Python?

In Python, the pass statement is a null operation, meaning it does nothing when executed. It is often used as a placeholder where syntactically a statement is required but no action needs to be taken. The pass statement is primarily used for maintaining code structure, indicating intentional absence of functionality, or as a placeholder during development.

What is Polymorphism?

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different underlying forms or types.

What is operator overloading in Python?

Operator overloading in Python refers to the ability to define how operators behave for custom objects. Python allows you to redefine the behaviour of built-in operators such as +, -, *, /, ==, <, >, and others, for objects of user-defined classes.

Key Points about Operator Overloading:

1. Customizing Operator Behavior:

By overloading operators, you can define custom behavior for operations involving objects of your own classes.

2. Special Method Names:
   Operator overloading is achieved in Python by implementing special methods with predefined names, also known as magic methods or dunder methods (due to the double underscores in their names).
   Example of Operator Overloading:
   For example, defining the __add__() method allows you to specify how instances of your class should behave when the + operator is used with them.

What different types of inheritance are there?

In object-oriented programming, inheritance is a fundamental concept where a class (known as a child or subclass) derives properties and behaviour (methods) from another class (known as a parent or superclass). Python supports several types of inheritance, each with its own characteristics and use cases:

1. Single Inheritance: A subclass inherits from one superclass.
2. Multiple Inheritance: A subclass inherits from more than one superclass.
3. Multilevel Inheritance: A class inherits from a superclass, and another class inherits from this derived class, forming a chain.
4. Hierarchical Inheritance: Multiple subclasses inherit from a single superclass.
5. Hybrid Inheritance: A combination of two or more types of inheritance.

What is a superclass?

A superclass, also known as a base class or parent class, is a class from which other classes (subclasses or derived classes) inherit properties and behaviours. Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit attributes and methods from another class. The superclass provides a common interface and shared behaviour that can be extended or overridden by subclasses.

What is the use of the super() function?

The super() function in Python is used to give access to methods and properties of a parent or sibling class. It returns a temporary object of the superclass that allows you to call its methods. This is especially useful in the context of inheritance, where a subclass needs to extend or modify the behaviour of its superclass without duplicating code.

Regular Expressions

Theory/Conceptual based

What is a regular expression?

A regular expression, often abbreviated as regex or regexp, is a sequence of characters that define a search pattern. It is used for pattern matching within strings, allowing for powerful text processing capabilities. Regular expressions are commonly used in programming, text processing, and data validation.

How do you use the re module in Python?

The re module in Python provides support for working with regular expressions. It includes functions for searching, matching, and manipulating strings using regex patterns. The re module in Python is powerful for text processing tasks that involve pattern matching, searching, and manipulation.

Explain the difference between match() and search() in regular expressions.

	re.match()	**re.search()**
Purpose	The re.match() function attempts to match a pattern at the beginning of the string.	The re.search() function searches the entire string for the first occurrence of the pattern.
Behaviour	If the pattern is found at the start of the string, re.match() returns a match object. If the pattern is not at the start of the string, it returns None.	If the pattern is found anywhere in the string, re.search() returns a match object. If the pattern is not found, it returns None.
Use Case	Use re.match when you want to ensure that the string starts with a specific pattern.	Use re.search() when you want to find the pattern anywhere in the string.

What is the purpose of the findall() method in regular expressions?

The findall() method in Python's re module is used to search a string for all occurrences that match a specified regular expression pattern. It returns a list of all matches found in the string. This method is particularly useful when you want to extract all matching substrings at once.

How can you use groups in regular expressions?

Groups in regular expressions allow you to capture and extract specific parts of a matched pattern by enclosing them within parentheses, enabling more precise pattern matching and extraction. Once a pattern with groups matches a string, the text matched by each group can be accessed separately using methods like group() or groups(), providing structured access to relevant portions of the matched text. This facilitates focused extraction and streamlined processing of matched text components, enhancing the effectiveness of tasks such as data parsing, validation, and extraction in text processing applications.

How do you find all occurrences of a pattern in a string?

To find all occurrences of a pattern in a string, you can use the findall() method from the `re` module in Python. This method searches the entire string for all non-overlapping matches of the specified regular expression pattern and returns them as a list of strings. Each string in the list represents a separate occurrence of the pattern found in the input string. This approach is particularly useful when you need to extract multiple occurrences of a pattern from a string simultaneously.

What does the re.sub() function do?

The re.sub() function in Python's re module is used for replacing occurrences of a pattern in a string with a specified replacement string. It searches the input string for all occurrences of the specified regular expression pattern and replaces them with the replacement string. The function returns a new string where all replacements have been made. This method is particularly useful for performing text transformations, substitutions, or cleaning operations on strings based on matching patterns.

Explain the use of character classes in regular expressions with an example.

Character classes in regular expressions allow you to specify a set of characters that can match at a particular position in the string. They are denoted by enclosing the set of characters within square brackets [...]. Character classes offer a concise way to match any single character from the specified set.

Example:
Suppose you want to match any vowel (a, e, i, o, u) in a string. You can use a character class to achieve this:

```
import re
pattern = r'[aeiou]'
string = "Hello, how are you?"
matches = re.findall(pattern, string)
print("Matches:", matches)
```

Output: Matches: ['e', 'o', 'o', 'a', 'e', 'o', 'u']

What is a capturing group in regular expressions?

A capturing group in regular expressions is a part of a pattern that is enclosed within parentheses `( )`. It serves two main purposes:

1. Grouping: It allows you to group multiple characters, metacharacters, or other regular expressions together as a single unit within the pattern.
2. Capturing: It captures and remembers the text matched by the enclosed part of the pattern during a match operation, allowing you to extract specific portions of the matched text.

How can you use named groups in Python regular expressions?

In Python regular expressions, named groups provide a way to assign a name to capturing groups in addition to their numeric group number. This allows for more descriptive and readable code, especially when working with complex patterns. Named groups are defined using the syntax (?P<name>pattern), where name is the name of the group and pattern is the regular expression pattern enclosed within parentheses.

How do you perform a non-greedy match in regular expressions?

In regular expressions, a non-greedy match (also known as lazy or minimal match) is a pattern matching behaviour where the regex engine matches as little text as possible while still allowing the overall pattern to match successfully. This is achieved by adding a ? quantifier after a greedy quantifier (*, +, {m,n}). The ? makes the preceding quantifier non-greedy. Non-greedy matches are useful when you want to avoid matching too much text in cases where there are multiple occurrences of the pattern in the string. They help in making the pattern more precise and avoiding unintended matches.

What is the purpose of the re.IGNORECASE flag?

The re.IGNORECASE flag, also known as re.I, is a flag used in Python regular expressions to perform a case-insensitive match. When this flag is enabled, the regex engine ignores the case distinctions while matching characters in the pattern against the input string. This means that uppercase and lowercase letters are treated as equivalent during the matching process. The purpose of the re.IGNORECASE flag is to make the pattern matching process insensitive to the case of the characters in the input string, thereby allowing patterns to match regardless of whether the characters are in uppercase or lowercase.

Python Packages

What is a Python package and how do you install one?

A Python package is a collection of modules and sub-packages that provide reusable functions, classes, and variables. You can install a package using the pip command, such as:

```
pip install package_name
```

What is the difference between a Python package and a Python module?

A Python module is a single file containing Python code, such as functions, classes, and variables, typically saved with a .py extension. In contrast, a Python package is a collection of related modules organised in a directory hierarchy and includes a special __init__.py file to indicate that the directory is a package. While modules are imported using the import statement directly referencing the .py file, packages allow for a more organised and hierarchical structure, enabling the import of sub-modules or sub-packages using dot notation.

What is NumPy and why is it useful?

NumPy is a powerful numerical computing library in Python that provides support for arrays, matrices, and many mathematical functions. It is useful because it allows for efficient storage and manipulation of large datasets, and its operations are implemented in C, which makes them very fast.

Explain the difference between a list and a NumPy array.

A list is a built-in Python data type that can hold heterogeneous elements, whereas a NumPy array is a homogeneous data structure from the NumPy library optimised for numerical operations. NumPy arrays support vectorized operations, which means you can perform element-wise operations efficiently without explicit loops.

What is pandas and what are it's primary use cases?

Pandas is a data manipulation and analysis library. It is primarily used for data cleaning, transformation, and analysis. The main data structures are Series (1-dimensional) and DataFrame (2-dimensional).

What is the difference between loc and iloc in pandas?

'loc' is label-based indexing, meaning you select rows and columns by their labels. 'iloc' is integer-based indexing, meaning you select rows and columns by their integer position.

What is the difference between DataFrame and Series in pandas?

A Series is a one-dimensional labelled array capable of holding any data type, similar to a column in a table. A DataFrame is a two-dimensional labelled data structure with columns of potentially different types, similar to a table or spreadsheet.

What is a pivot table in pandas and why is it useful?

A pivot table in pandas is a data summarization tool that allows you to aggregate and reorganise data to gain insights. It's useful for transforming and analysing data, summarising information by categories, and performing operations like mean, sum, count, etc., on grouped data.

What is the difference between pivot and pivot_table in pandas?

The pivot method is used to reshape data (create a new DataFrame) based on column values and does not allow for aggregation. pivot_table, on the other hand, allows for aggregation of data using functions like sum, mean, etc., and is more flexible, handling duplicate entries by applying the specified aggregation function.