################################
OOPs Concepts in Python
################################

Video summary [00:00:00][1] - [00:11:34][2]:
Part 1 of the video talks about the basics of object-oriented programming (OOP) in Python. It covers the concepts of objects, classes, and data types, and how they are used to create real-life applications.
Highlights:
- [00:00:00][3] What is OOP and why it is important
- OOP is a different style of writing code
- OOP helps to build software that solves real-life problems
- OOP allows to create custom data types
- [00:05:27][4] What is an object and a class
- An object is anything that has some properties and behaviour
- An object is always an instance of a class
- A class is a blueprint or template for creating objects
- [00:09:34][5] What are the core features of OOP
- Abstraction, encapsulation, inheritance, and polymorphism
- These features help to make the code more organized, reusable, and flexible
- [00:10:18][6] What is a class and how to define it
- A class is defined using the keyword class and a name
- A class has attributes and methods
- Attributes are variables that store data
-Methods are functions that perform actions

Video summary [00:11:37][1] - [00:24:33][2]:
Part 2 of the video talks about the concept of classes and objects in Python. It covers the definition, syntax, and examples of creating and using classes and objects. It also explains the difference between data and methods, and how to represent classes diagrammatically.
Highlights:
- [00:11:37][3] What is a class and an object
- A class is a blueprint of how an object will behave
- An object is an instance of a class
- A data type is a class and a variable is an object
- [00:14:23][4] How to create a class in Python
- Use the keyword class followed by the name of the class
- Use title case for the class name and snake case for the variables and functions
- Define the attributes (data) and methods (functions) inside the class
- [00:17:28][5] How to create an object in Python
- Use the class name followed by parentheses
- Assign the object to a variable
- Access the attributes and methods using the dot operator
- [00:20:01][6] How to represent a class diagrammatically
- Use a rectangle with three sections
- Write the class name in the top section
- Write the attributes in the middle section
- Write the methods in the bottom section
-Use plus and minus signs to indicate public and private access

---

Video summary [00:24:36][1] - [00:37:55][2]:
Part 3 of the video talks about the difference between functions and methods in Python, and how to create a class for an ATM machine. It covers the concepts of constructor, self, and input, and how to define and call methods inside a class.
Highlights:
- [00:24:36][3] The difference between functions and methods
- Functions are defined outside a class and can be used by anyone
- Methods are defined inside a class and can only be accessed by the class object
- Methods are called with a dot notation, such as list.append()
- [00:27:03][4] The constructor method
- The constructor is a special method that is automatically executed when a class object is created
- The constructor name is always init in Python
- The constructor can take arguments and assign them to the class attributes
- [00:29:58][5] The self parameter
- Self is a reference to the current class object
- Self is always the first parameter of any method in a class
- Self is used to access the class attributes and methods
- [00:31:07][6] The input function
- Input is a built-in function that takes user input from the keyboard
- Input can take a string argument as a prompt message
- Input always returns a string value
- [00:32:26][7] The ATM class
- The ATM class is an example of object-oriented programming in Python
- The ATM class has two attributes: pin and balance
-The ATM class has four methods: create_pin, deposit, withdraw, and check_balance

---

Video summary [00:37:58]1 - [00:52:15]2:
Part 4 of the video talks about how to create a basic ATM software in Python using object-oriented programming (OOP). It covers the concepts of classes, objects, methods, inheritance, and magic methods.
Highlights:
• [00:37:58]3 Creating a class for bank accounts Defines the attributes and methods of a bank account
• Uses the constructor method to initialize the account details
• Uses the deposit and withdraw methods to update the balance

• [00:41:07]4 Creating a subclass for SBI accounts Inherits the attributes and methods of the bank account class
• Overrides the deposit and withdraw methods to add extra features
• Uses the check balance method to display the balance

• [00:45:02]5 Understanding magic methods Explains that magic methods are special methods that start and end with double underscores
• Shows that magic methods are automatically triggered by certain actions or operators
• Gives examples of magic methods such as init, str, and add

• [00:49:04]6 Using the ATM software Creates objects for different bank accounts using the classes

- Calls the methods on the objects to perform various operations
- Demonstrates the use of magic methods to print and add the objects

---

Video summary [00:52:18][1] - [01:05:10][2]:
Part 5 of the video talks about the concept of self in Python, which is a special parameter that refers to the current object. It explains how self is used to access the data and methods of a class, and how it is passed by default when an object calls a method. It also shows how to use self to create constructors, which are special methods that are executed automatically when an object is created.
Highlights:
- [00:52:18][3] The purpose of constructors
- They are used to initialize the data and configuration of an object
- They are not controlled by the user, but by the program
- They are defined with the special method __init__
- [00:55:37][4] The meaning of self
- It is a reference to the current object
- It is passed as the first argument to every method of a class
- It is used to access the data and methods of a class
- [00:59:20][5] The rule of self
- A class's method can only access its own data and methods through self
- A class's method cannot access another method of the same class directly
- The power to access a class's method is only with the object of that class
- [01:03:04][6] The need of self
- A class can have only two things: data and methods
- Only the object of a class can access its data and methods
- Self is needed to identify which object is calling a method

---

Video summary [01:05:12][1] - [01:17:24][2]:
Part 6 of the video talks about how to create and use a custom class for fractions in Python. It covers the concepts of class attributes, methods, constructors, and special methods. It also shows how to implement arithmetic operations on fraction objects using operator overloading.
Highlights:
- [01:05:12][3] The need for a custom class
- Data and functions are encapsulated in a class
- Only the class can access its own data and functions
- Sometimes, different classes need to interact with each other
- [01:06:12][4] The creation of a fraction class
- Define the class name as Fraction
- Define the constructor method to take two arguments: numerator and denominator
- Assign the arguments to the class attributes: self.numerator and self.denominator
- [01:09:12][5] The printing of a fraction object
- Use the special method str to define how the object is printed
- Return a string that contains the numerator and denominator separated by a slash
- Use string formatting to insert the values of the attributes

- [01:12:12][6] The addition of two fraction objects
- Use the special method add to define how the object is added to another object
- Take two arguments: self and other
- Calculate the new numerator and denominator using the formula for fraction addition
- Return a new fraction object with the calculated values

Video summary [01:17:27][1] - [01:30:13][2]:
Part 7 of the video talks about how to create a fraction class in Python using object-oriented programming
. It covers the concepts of instance variables, magic methods, and data hiding.
Highlights:
- [01:17:27][3] How to define a fraction class with a constructor
- Use the init method to initialize the numerator and denominator
- Use the str method to return a string representation of the fraction
- [01:18:40][4] How to implement arithmetic operations using magic methods
- Use the add method to add two fractions
- Use the sub method to subtract two fractions
- Use the mul method to multiply two fractions
- Use the truediv method to divide two fractions
- [01:20:29][5] How to convert a fraction to a float using a property
- Use the @property decorator to define a method that returns the float value of the fraction
- Use the float method to call the property method
- [01:21:05][6] How to use other magic methods for different purposes
- Use the repr method to return a string that can be used to recreate the object
- Use the hex, oct, and complex methods to convert the fraction to different types
- Use the int method to convert the fraction to an integer
- [01:22:03][7] How to hide the data of the class using double underscore
- Use the double underscore prefix to make the instance variables private
- Use the double underscore prefix to access the private variables inside the class
- Use the double underscore prefix to prevent the outside access and modification of the data

---

Video summary [01:30:18][1] - [01:44:23][2]:
Part 8 of the video talks about how to use double underscore to hide data members and methods in Python classes. It explains the concept of name mangling and how it can be used to access private attributes. It also shows how to create getter and setter functions to control the access and modification of data members.
Highlights:
- [01:30:18][3] Using double underscore to hide data members
- Prefixes data members with double underscore to make them private
- Replaces the name with _ClassName__DataMember internally
- Prevents direct access from outside the class
- [01:31:27][4] Using double underscore to hide methods

- Prefixes methods with double underscore to make them private
- Replaces the name with _ClassName__Method internally
- Prevents direct access from outside the class
- [01:32:31][5] Using name mangling to access private attributes
- Uses the _ClassName__Attribute syntax to access private attributes
- Shows an example of accessing and modifying the private pin attribute
- Warns about the potential problems of name mangling
- [01:38:27][6] Creating getter and setter functions for data members
- Creates a get_pin function to return the value of the private pin attribute
- Creates a set_pin function to change the value of the private pin attribute
- Checks the type and value of the new pin before setting it
- Prints a message when the pin is changed

Video summary [01:44:26][1] - [01:58:34][2]:
Part 9 of the video talks about the concept of encapsulation in object-oriented programming in Python. It covers the need for encapsulation, the use of getters and setters, the class diagram notation, and the reference variable.
Highlights:
- [01:44:26][3] The need for encapsulation
- To protect data from direct access or modification
- To hide data using double underscore prefix
- To provide access and update using getters and setters
- [01:46:14][4] The use of getters and setters
- To create methods for each data member
- To return and set the value of the data member
- To apply logic or validation in the methods
- [01:46:26][5] The class diagram notation
- To represent the class name, attributes, and methods
- To use symbols for public (+), private (-), and protected (#)
- To use the class name as the reference variable
- [01:49:00][6] The reference variable
- To store the address of the object in memory
- To access the object’s attributes and methods
- To pass the object as an argument to a function

Video summary [01:58:36][1] - [02:14:55][2]:
Part 10 of the video talks about object-oriented programming in Python. It covers the concepts of objects,classes, constructors, methods, inheritance, and polymorphism. It also explains the difference between mutable and immutable objects, and how to pass objects as arguments to functions.
Highlights:
- [01:58:36][3] Objects and classes
- An object is an instance of a class
- A class defines the attributes and methods of an object
- A constructor is a special method that initializes an object
- [02:00:17][4] Passing objects to functions
- An object can be passed as an argument to a function
- The function can access and modify the attributes and methods of the object
- The changes are reflected in the original object
- [02:04:00][5] Mutable and immutable objects

- A mutable object can be changed after it is created
- A list is an example of a mutable object
- An immutable object cannot be changed after it is created
- A tuple is an example of an immutable object
- [02:08:02][6] Inheritance and polymorphism
- Inheritance is the mechanism of creating a new class from an existing class
- The new class inherits the attributes and methods of the existing class
- Polymorphism is the ability of an object to behave differently depending on the context
- The same method can have different implementations in different classes

---

Video summary [02:14:57][1] - [02:30:01][2]:
Part 11 of the video talks about how to implement object-oriented programming in Python. It covers the concepts of class variables, static methods, class methods, and inheritance. It also shows how to use the @classmethod and @staticmethod decorators to define different types of methods.
Highlights:
- [02:14:57][3] Class variables
- Variables that are shared by all instances of a class
- Can be accessed by self or class name
- Can be updated by class name
- [02:17:18][4] Static methods
- Methods that do not need self or class as arguments
- Used for operations that do not depend on instance or class attributes
- Marked by @staticmethod decorator
- [02:20:13][5] Class methods
- Methods that take class as the first argument
- Used for operations that depend on class attributes
- Can be used to create alternative constructors
- Marked by @classmethod decorator
- [02:25:21][6] Inheritance
- The process of creating a new class from an existing class
- The new class inherits the attributes and methods of the parent class
- The new class can override or extend the parent class’s functionality
- The parent class can be accessed by super() function

Video summary [02:30:10][1] - [02:45:39][2]:
Part 12 of the video talks about the concept of aggregation and inheritance in object-oriented programming in Python. It explains the difference between these two types of relationships between classes, and gives examples of how to implement them using constructors, methods, and attributes.
Highlights:
- [02:30:10][3] The concept of aggregation
- A type of relationship where one class contains another class as an attribute
- Also called has-a relationship
- Example: A customer class has an address class
- [02:34:01][4] The concept of inheritance

- A type of relationship where one class inherits the properties and methods of another class
- Also called is-a relationship
- Example: A smartphone class is a product class
- [02:39:30][5] The benefits of inheritance
- Allows code reuse and avoids duplication
- Enables polymorphism and dynamic binding
- Example: A student class and an instructor class inherit from a user class
- [02:41:54][6] The implementation of inheritance
- Use the super() function to call the parent class constructor
- Override or extend the parent class methods and attributes
- Example: An edit profile method for the student class and the instructor class

---

Video summary [02:45:41][1] - [02:57:42][2]:
Part 13 of the video talks about how to use inheritance in object-oriented programming in Python. It covers the concept, syntax, and benefits of inheritance, and shows an example of a user class and a student class that inherits from it.
Highlights:
- [02:45:41][3] The concept of inheritance
- Allows a child class to inherit properties and methods from a parent class
- Reduces code repetition and increases code reusability
- Follows the DRY principle: Don’t Repeat Yourself
- [02:48:23][4] The syntax of inheritance
- Use parentheses after the child class name to specify the parent class
- Use super() to access the parent class constructor and methods
- Private members are not inherited
- [02:51:28][5] The benefits of inheritance
- Makes the code more organized and modular
- Allows polymorphism: the ability to use a child class object as a parent class object
- Enhances code readability and maintainability
- [02:55:32][6] An example of inheritance
- Defines a user class with login and register methods
- Defines a student class that inherits from the user class
- Adds enroll and review methods to the student class
- Shows how the student object can access both the user and student methods

---

Video summary [02:57:45][1] - [03:09:14][2]:
Part 14 of the video talks about object-oriented programming in Python, covering the concepts of inheritance, method overriding, and polymorphism. It uses examples of classes and objects to illustrate how these concepts work in practice.
Highlights:
- [02:57:45][3] Inheritance
- Allows a child class to inherit attributes and methods from a parent class
- The child class can access the public attributes of the parent class
- The child class can also override the methods of the parent class
- [03:02:31][4] Method overriding

- Occurs when a child class defines a method with the same name and parameters as a parent class
- Allows the child class to customize the behaviour of the method
- The method of the child class will be executed instead of the parent class
- [03:05:58][5] Polymorphism
- Refers to the ability of an object to behave differently depending on its type
- Allows the same method name to have different implementations in different classes
- Enables code reuse and flexibility

Video summary [03:09:17][1] - [03:21:15][2]:
Part 15 of the video talks about the use of super keyword in Python. It covers the concept of method overriding, the syntax and purpose of super, and some examples of how to invoke parent class methods and at tributes from child class.
Highlights:
- [03:09:17][3] The concept of method overriding
- When a child class has a method with the same name as the parent class
- The child class method will execute instead of the parent class method
- [03:10:25][4] The syntax and purpose of super
- super() is a built-in function that returns a proxy object of the parent class
- super() can be used to access the parent class methods and attributes from the child class
- super() can avoid code duplication and improve readability
- [03:12:11][5] An example of using super to invoke parent class constructor
- A Phone class with attributes price, brand, and camera
- A Smartphone class that inherits from Phone and has additional attributes os and ram
- super() is used to pass the common attributes to the parent class constructor
- [03:14:47][6] An example of using super to invoke parent class method
- A Parent class with a show() method that prints the name attribute
- A Child class that inherits from Parent and has a show() method that prints the val attribute
- super() is used to call the parent class show() method from the child class show() method

---

Video summary [03:21:17][1] - [03:34:17][2]:
Part 16 of the video talks about the types of inheritance in Python, which are single, multilevel, hierarchical, multiple, and hybrid. The video explains the concept of method resolution order (MRO) and how it determines which parent class's method is executed in case of multiple inheritance. The video also shows examples of each type of inheritance using classes such as Phone, Product, Smartphone, Teacher, etc.
Highlights:
- [03:22:16][3] Single level inheritance
- A child class inherits from one parent class
- The child class can access the parent class's methods and properties
- [03:23:27][4] Multilevel inheritance

- A child class inherits from a parent class, which in turn inherits from another parent class
- The child class can access the methods and properties of both the parent class and the grandparent c
lass
- [03:24:42][5] Hierarchical inheritance
- A single parent class has multiple child classes that inherit from it
- Each child class can access the parent class's methods and properties
- [03:25:07][6] Multiple inheritance
- A child class inherits from more than one parent class
- The child class can access the methods and properties of all the parent classes
- The MRO decides which parent class's method is executed first in case of conflict
- [03:25:10][7] Hybrid inheritance
- A combination of any of the above types of inheritance
- The MRO also applies to hybrid inheritance

Video summary [03:34:20][1] - [03:46:01][2]:
Part 17 of the video talks about method overloading and operator overloading in Python. It explains the concepts and differences of these two types of polymorphism, and shows how to implement them using magic methods. It also gives an example of a small project that uses the Indian Rail API to perform various operations related to train information.
Highlights:
- [03:34:20][3] Method overloading
- Allows a method to have different behaviours based on different inputs
- Not supported natively in Python, but can be achieved using default arguments or if-else logic
- Example: area() method for different shapes
- [03:35:25][4] Operator overloading
- Allows operators to have different behaviours based on different operands
- Can be implemented using magic methods that start and end with double underscores
- Example: + operator for addition or concatenation
- [03:42:15][5] Magic methods
- Special methods that are invoked automatically when certain operations are
performed
- Can be used to customize the behaviour of built-in operators and functions
- Example: add() for +, str() for str(), len() for len()
- [03:44:24][6] Small project
- A mini application that uses the Indian Rail API to get train information
- Uses requests module to send HTTP requests and get JSON responses
- Demonstrates various functions such as train schedule, live status, seat
availability, etc.

Video summary [03:46:04][1] - [03:58:17][2]:
Part 18 of the video talks about how to use Python to fetch data from an API and display the train schedule information. It covers the steps of installing the requests library, making a GET request, parsing the JSON data, and looping through the dictionary items.
Highlights:
- [03:46:04][3] How to create an account and login to the API website
- Shows the URL and the API key
- Copies and pastes the URL in the browser
- Enters a train number to check the live status
- [03:47:26][4] How to use JSON viewer to see the data structure
- Copies and pastes the JSON data in the tool
- Explains the dictionary and list items
- Shows the root key and the station name values
- [03:49:11][5] How to install the requests library in PyCharm
- Uses the command pip install requests
- Imports the requests module
- Uses the get function to make a request
- [03:50:11][6] How to parse the JSON data and convert it to a dictionary
- Uses the json method to get the data
- Prints the data and the type
- Formats the URL with the train number
- [03:51:57][7] How to loop through the dictionary and get the station names
- Uses a for loop to iterate over the data
- Prints the i item and the type
- Accesses the station name key inside the i item

Video summary [03:58:20][1] - [04:02:55][2]:
Part 19 of the video talks about how to create a web application in Python using an API. It covers the steps of setting up the API key, making requests, and parsing the data.
Highlights:
- [03:58:20][3] How to print the train details using the Train class
- Shows an example of printing the name, arrival time, departure time, and distance of a train
- Uses string formatting and f-strings to display the information
- [04:00:06][4] How to use the Indian Railway API to get the train status
- Explains the requirements and parameters of the API
- Provides an example of the API key and the station code
- Shows how to format the date and time for the API request
- [04:01:03][5] How to make a request to the API and parse the data
- Imports the requests and json modules
- Uses the requests.get() method to send the request
- Uses the json.loads() method to convert the response to a dictionary
- Accesses the relevant data from the dictionary
- [04:02:31][6] How to handle the errors and limitations of the API
- Explains the possible reasons for the API not working
- Suggests to try with a different account or a different website
- Advises to check the documentation and the data format of the API