

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Course Work**

**Topic:** Object Oriented Design – Implement Simple ATM

**Department**: Computer system and programming

**Faculty:** Aerospace

**Course:** Ⅲ

**Speciality:** Computer engineering

**Group:** 2450i

**Subject:** Object Oriented Programming

**Student:** Shirinov Turan

**Head of deparment:**  Məlikov Ağasi

**Supervisor:** Sahmaliyev Memmed

**Mark:**

**Baku 2022**

# **Content**

**1.Python**

**1.1 Programming language ----------------------------------**

**1.2 Why Python? --------------------------------------------**

**2.OOP**

**2.1 OOP concept ------------------------------------------------**

**2.2 OOP in Python ----------------------------------------------**

**2.3 Classes -------------------------------------------------------**

**3.Implementing ATM with classes**

**3.1 Basic idea of ATM -------------------------------------------------**

**3.2 Classes of ATM -----------------------------------------------**

**4.Class explanation**

**4.1 Explanation of each class -----------------------------------**

**4.2 Attributes of classes ---------------------------------------**

**5. Examples**

**5.1 Action based on classes ------------------------------------------**

**6.Database – MySQL**

**6.1 What is Database? -----------------------------------------**

**6.2 Importance of database (MySQL) ----------------------**

**6.3 Connection between DB and Python --------------------**

**References -----------------------------------------------------------**

# **1.Python**

**Programming language**

Programming languages are the building blocks of modern technology. They allow us to communicate with computers and create complex software applications that power the world around us. A programming language is a set of rules and syntax that define how a computer program is written and executed. There are many programming languages available today, each with its own strengths and weaknesses.

One of the most popular programming languages is Java. Java is a powerful, object-oriented language that is used in many different types of applications, from web applications to mobile apps. It is known for its platform independence, meaning that Java code can be run on any device that has a Java Virtual Machine (JVM) installed. Java is also highly secure, making it a popular choice for applications that handle sensitive data.

Another popular language is Python. Python is a high-level language that is known for its readability and simplicity. It is used in a wide range of applications, from scientific computing to web development. Python's syntax is straightforward and easy to learn, making it a great language for beginners. It also has a large and active community, which means there are plenty of resources and libraries available for developers to use.

C++ is another powerful programming language that is widely used in the development of operating systems, video games, and other high-performance applications. It is an object-oriented language that is known for its speed and efficiency. C++ is a complex language that can be challenging to learn, but it is also highly versatile and can be used to create a wide range of applications.

JavaScript is a popular language that is used in web development to create interactive and dynamic web pages. It is a client-side language, which means that it is executed by the user's web browser. JavaScript is known for its flexibility and ease of use, and it is a great choice for creating rich, interactive web applications.

In conclusion, programming languages are an essential part of modern technology. Whether you are building a mobile app, a web application, or an operating system, there is a programming language that is right for the job. Each language has its own unique features and strengths, and it is up to the developer to choose the language that is best suited for their project.

**Why Python?**

Python is an interpreted language, meaning that it does not need to be compiled before running. This makes it a great language for beginners, as it eliminates the need for complex build processes. The language's syntax is straightforward and easy to understand, which helps developers write code more quickly and efficiently. In addition, Python has a large and active community of developers, which means that there is a wealth of resources and libraries available to help developers solve complex problems.

One of the key strengths of Python is its versatility. It can be used for a wide range of applications, from scientific computing to web development. Python's versatility comes from its vast array of libraries, which can be easily imported into projects. These libraries provide a range of functionalities, such as data analysis, machine learning, and web development.

Another significant advantage of Python is its focus on readability. Python's syntax is designed to be as clear and concise as possible, making it easy to read and understand. This means that Python code can be easily maintained and modified, even by developers who are unfamiliar with the project.

Python is also known for its extensive documentation, which is available online and is continuously updated by the community. This documentation provides detailed explanations of the language's syntax, libraries, and functionalities, making it easy for developers to get started with Python.

In conclusion, Python is a powerful and versatile programming language that has become increasingly popular in recent years. Its simplicity, versatility, and ease of use make it an excellent language for beginners, while its vast array of libraries and functionalities make it a popular choice for more advanced projects. Python's focus on readability and extensive documentation also make it an excellent choice for large-scale projects that require frequent modifications and maintenance. Whether you are a beginner or an experienced developer, Python is a language that is definitely worth learning.

**2.OOP**

**OOP concept**

Object-oriented programming (OOP) is a programming paradigm that is based on the concept of objects, which are instances of classes. In OOP, a class is a blueprint or template for creating objects, which encapsulate data and behavior into a single entity.

The fundamental principles of OOP include encapsulation, inheritance, and polymorphism. Encapsulation refers to the practice of hiding implementation details inside an object, allowing only the necessary information to be accessed from the outside. This helps to prevent unintended interference and ensures that code remains organized and maintainable.

Inheritance is the ability for a class to inherit properties and methods from another class. This allows developers to create hierarchies of classes that share common features, reducing redundancy and improving code organization. Inheritance also promotes code reuse, as common functionality can be abstracted into a superclass and shared across multiple subclasses.

Polymorphism refers to the ability for objects to take on multiple forms or types. This allows developers to create flexible and dynamic code that can be adapted to different scenarios. Polymorphism is achieved through the use of interfaces, which define a set of methods that must be implemented by any class that implements the interface. This allows different objects to be treated as interchangeable, as long as they adhere to the same interface.

OOP has many benefits, including improved code organization, increased code reuse, and enhanced maintainability. It also promotes modularity and flexibility, as objects can be easily added, removed, or modified without affecting other parts of the code. Additionally, OOP provides a more intuitive way of modeling real-world concepts and relationships, making it easier for developers to understand and work with complex systems.

In conclusion, OOP is a powerful programming paradigm that is based on the concept of objects. It provides a range of benefits, including improved code organization, increased code reuse, and enhanced maintainability. By leveraging encapsulation, inheritance, and polymorphism, developers can create flexible, modular, and scalable code that can be adapted to a wide range of scenarios.

**OOP in Python**

Python is a popular and versatile programming language that supports multiple programming paradigms, including object-oriented programming (OOP). In Python, OOP is implemented through the use of classes, which are templates for creating objects that encapsulate data and behavior into a single entity.

One of the key benefits of OOP in Python is the ability to create reusable code through the use of inheritance. Inheritance allows classes to inherit properties and methods from a parent or base class, which can then be modified or extended in the child class. This promotes code reuse and reduces redundancy, as common functionality can be abstracted into a superclass and shared across multiple subclasses.

Another key aspect of OOP in Python is encapsulation, which involves hiding implementation details inside an object and exposing only the necessary information to the outside world. This helps to prevent unintended interference and ensures that code remains organized and maintainable. Encapsulation is achieved in Python through the use of access modifiers such as public, private, and protected, which control the visibility of class members.

Polymorphism is another important aspect of OOP in Python, which allows objects to take on multiple forms or types. This is achieved through the use of inheritance and interfaces, which define a set of methods that must be implemented by any class that implements the interface. This allows different objects to be treated as interchangeable, as long as they adhere to the same interface.

In Python, OOP is also used extensively in the creation of GUI applications, web applications, and game development. GUI libraries such as PyQt and Tkinter provide classes that can be extended and customized to create interactive and responsive user interfaces. Similarly, web frameworks such as Django and Flask use OOP to create modular and scalable web applications. Finally, game development libraries such as Pygame and PyOpenGL use OOP to create immersive and engaging games.

Python's OOP features are easy to use and understand, making it an ideal language for beginners as well as experienced developers. Python's syntax is concise and readable, which makes it easy to create and work with classes and objects. Additionally, Python's dynamic typing system allows for flexible and dynamic code that can be easily adapted to changing requirements.

In conclusion, OOP is a powerful programming paradigm that is supported by Python. In Python, OOP is implemented through the use of classes, which are templates for creating objects that encapsulate data and behavior. OOP in Python provides a range of benefits, including code reuse, encapsulation, and polymorphism. It is widely used in the creation of GUI applications, web applications, and game development, making it a versatile and useful programming paradigm for a wide range of applications.

**Classes**

In OOP, classes are a fundamental concept that allows developers to create complex, organized, and reusable code. A class acts as a blueprint or template that defines the structure and behavior of an object. It specifies the attributes, or properties, of the object, such as its name, size, or color, as well as the methods, or functions, that define its behavior, such as how it interacts with other objects or how it performs calculations.

Classes provide a way to model real-world objects and concepts in code, and they are used extensively in the creation of software applications. For example, a class might represent a car, with properties such as its make, model, and year, and methods such as accelerate, brake, and turn. A class could also represent a user in a system, with properties such as their name, email address, and password, and methods such as login and logout.

One of the key benefits of classes is that they allow for code reusability. Once a class has been defined, it can be instantiated multiple times to create individual objects with their own unique properties and behaviors. This saves developers time and effort by eliminating the need to write redundant code.

In addition, classes promote encapsulation, which means that the internal workings of an object are hidden from the outside world. This prevents unintended interference and ensures that code remains organized and maintainable. Classes achieve encapsulation by using access modifiers, which control the visibility of class members such as properties and methods.

Overall, classes are a powerful tool in OOP that allow developers to create complex, modular, and reusable code. They provide a way to model real-world objects and concepts in code, and they promote code reusability and encapsulation. By mastering the concept of classes, developers can create robust and scalable software applications.

**3.Implementing ATM with classes**

**Basic idea of ATM**

An automated teller machine (ATM) is a device that allows individuals to perform a variety of banking transactions without the need for a human teller. ATMs are widely used by individuals and businesses to withdraw cash, check their account balances, transfer money between accounts, and perform other banking services.

The basic idea of an ATM is to provide users with a convenient and secure way to access their bank accounts 24 hours a day, 7 days a week. An ATM typically consists of a computerized system that is connected to the user's bank account, a cash dispenser, a keypad or touch screen, and a card reader.

When a user approaches an ATM, they insert their debit or credit card into the card reader. The card reader reads the card's magnetic strip or chip and retrieves the user's account information from the bank's database. The user is then prompted to enter a personal identification number (PIN) to verify their identity.

Once the user has entered their PIN, they can then select the type of transaction they wish to perform, such as withdrawing cash or checking their account balance. The ATM will then communicate with the user's bank to complete the transaction. If the user is withdrawing cash, the ATM will dispense the requested amount of cash from the cash dispenser.

To ensure the security of the user's account information and transactions, ATMs use a variety of security measures. For example, the card reader may be designed to detect and prevent the use of counterfeit or stolen cards. The keypad or touch screen may be designed to prevent onlookers from seeing the user's PIN. And the ATM may be equipped with cameras to record any suspicious activity.

Overall, the basic idea of an ATM is to provide individuals with a convenient and secure way to access their bank accounts and perform a variety of banking transactions at any time of the day or night.

**Classes of ATM**

Our code defines three classes: Account, Bank, and ATM.

**4.Class explanation**

**Explanation of each class**

The `Account` class represents a bank account and has four attributes: `account\_number`, `balance`, `account\_holder`, and `pin`. It also has two methods: `withdraw()` and `deposit()`. The `withdraw()` method checks if the account has sufficient balance and then deducts the specified amount from the balance. The `deposit()` method adds the specified amount to the account balance.

The `Bank` class represents a bank and interacts with a MySQL database to store account details. It has three attributes: `db`, `cursor`, and `create\_table()`. The `db` attribute holds the connection to the MySQL database, the `cursor` attribute holds the cursor object for the database connection, and the `create\_table()` method creates a new table `accounts` in the database if it does not exist already. The `Bank` class also has several methods: `add\_account()`, `authenticate()`, `get\_balance()`, `withdraw()`, and `deposit()`. The `add\_account()` method inserts a new account into the `accounts` table in the database. The `authenticate()` method takes an account number and PIN as input and returns the corresponding `Account` object if the account number and PIN match with the stored values in the database. The `get\_balance()`, `withdraw()`, and `deposit()` methods interact with the database to update the account balance.

The `ATM` class represents an ATM machine and interacts with the `Bank` class to perform transactions. It has one attribute: `bank`. The `insert\_card()` method takes an account number and PIN as input, calls the `authenticate()` method of the `Bank` class, and starts the transaction menu if the authentication is successful. The `transaction\_menu()` method displays the transaction options to the user and calls the corresponding methods of the `ATM` and `Bank` classes based on the user's choice. The `balance\_inquiry()`, `withdrawal()`, and `deposit()` methods call the corresponding methods of the `Bank` class to retrieve or update the account balance.

**Attributes of classes**

1. Account class:

This class represents an account and has the following attributes:

- account\_number: a string representing the account number

- balance: a float representing the current balance of the account

- account\_holder: a string representing the name of the account holder

- pin: a string representing the PIN number of the account holder

The Account class has the following functions:

- \_\_init\_\_(): constructor function that initializes the attributes of the account object

- withdraw(): method that allows the account holder to withdraw money from the account, provided the balance is sufficient. Returns True if successful, False otherwise.

- deposit(): method that allows the account holder to deposit money into the account.

2. Bank class:

This class represents a bank and interacts with the database to create accounts, authenticate users, and perform account-related functions. The Bank class has the following functions:

- \_\_init\_\_(): constructor function that connects to the MySQL database, creates a cursor object and creates a table 'accounts' if it does not exist.

- create\_table(): method that creates the 'accounts' table in the database.

- add\_account(): method that adds a new account to the 'accounts' table.

- authenticate(): method that checks if the given account number and PIN are valid, and returns the account object if valid, None otherwise.

- get\_balance(): method that retrieves the balance of the given account.

- withdraw(): method that allows the account holder to withdraw money from the account, provided the balance is sufficient, and updates the database accordingly.

- deposit(): method that allows the account holder to deposit money into the account, and updates the database accordingly.

3. ATM class:

This class represents an ATM machine and provides a user interface for the account holder to access their account. The ATM class has the following functions:

- \_\_init\_\_(): constructor function that takes a Bank object as a parameter.

- insert\_card(): method that prompts the user to enter their account number and PIN, authenticates the user using the Bank object, and calls the transaction\_menu() method if authentication is successful.

- transaction\_menu(): method that displays a menu of options to the user and prompts them to choose an option. Depending on the user's choice, it calls the corresponding function from the Bank object.

- balance\_inquiry(): method that retrieves the balance of the given account and displays it to the user.

- withdrawal(): method that allows the account holder to withdraw money from the account, provided the balance is sufficient, and updates the database and displays a message to the user.

- deposit(): method that allows the account holder to deposit money into the account, updates the database and displays a message to the user.

**5. Examples**

**Action based on classes**

*The first code* for authenticating a user's account by asking for their account number and PIN, and then allowing them to access an ATM if the authentication is successful.

First, the code imports the `ATM` and `Bank` classes from a module called `atm`.

Then, an instance of the `Bank` class is created, and the user is prompted to enter their account number and PIN. The `authenticate` method of the `Bank` class is then called with the user's input, and the result is stored in the `authenticated\_account` variable.

If the authentication is successful (i.e., `authenticated\_account` is not `None`), the user is informed that the authentication was successful and an instance of the `ATM` class is created with the `Bank` instance passed as an argument. The `transaction\_menu` method of the `ATM` class is then called with `authenticated\_account` as an argument, which allows the user to access the menu for various ATM transactions.

If the authentication fails (i.e., `authenticated\_account` is `None`), the user is informed that the authentication failed and prompted to try again.

*The second code* allows a user to add a new bank account to the ATM system. It first imports the `Account` and `Bank` classes from the `atm` module, creates a new `Bank` object, and then prompts the user to enter details for the new account including the account number, initial balance, account holder's name, and account pin.

It then creates a new `Account` object with the input values and passes it to the `add\_account` method of the `Bank` object to add it to the database. Finally, it prints a message indicating that the account was added successfully.

**6.Database – MySQL**

**What is Database?**

A database is a structured collection of data stored and organized in a way that enables efficient data retrieval, manipulation, and management. It is a central component of most modern software applications and systems, ranging from web applications to enterprise resource planning (ERP) systems.

A database typically consists of one or more tables, which are collections of related data organized in rows and columns. Each row represents a single record or data point, while each column represents a data field or attribute. For example, a database of customer information might contain a table with columns for customer name, address, email, and phone number, with each row representing a single customer.

Databases can be broadly categorized into two main types: relational databases and non-relational databases. Relational databases are the most common type of database and are based on the relational model, which organizes data into tables with defined relationships between them. This relationship between tables is established using a primary key and foreign key constraints. Non-relational databases, on the other hand, store data in a more flexible format and do not have the rigid structure of tables and relationships found in relational databases. Examples of non-relational databases include document databases, graph databases, and key-value stores.

Databases are used for a wide range of applications and can be found in nearly every industry. In business, databases are used to store and manage customer data, financial records, and inventory information. In healthcare, databases are used to store and manage patient records, medical histories, and other health-related data. In scientific research, databases are used to store and analyze large volumes of data, such as genomic or astronomical data. In education, databases are used to store and manage student records, grades, and other educational data.

To interact with a database, a database management system (DBMS) is used. A DBMS is software that provides tools for creating, managing, and manipulating databases. Examples of popular DBMSs include MySQL, Oracle, Microsoft SQL Server, and MongoDB.

Databases are critical components of modern software systems and are essential for storing and managing large amounts of data efficiently. With the increasing volume and complexity of data being generated by businesses, individuals, and devices, the importance of databases will only continue to grow. As a result, understanding how databases work and how to use them effectively is becoming an increasingly important skill for software developers, data analysts, and other professionals working with data.

**Importance of database (MySQL)**

A database is an organized collection of data that can be accessed, managed and updated easily. It is used to store, manage and retrieve information in a structured way. Databases can be used for various purposes such as managing customer data, employee information, inventory, financial data, and much more.

One of the main benefits of using a database is that it allows users to easily access and manage large amounts of data. Without a database, managing large amounts of data can be time-consuming and prone to errors. Additionally, databases allow for multiple users to access the same data simultaneously, ensuring data consistency and avoiding data conflicts.

In the context of businesses, databases are particularly important. They allow businesses to efficiently store and manage customer data, which is crucial for delivering personalized customer experiences. Databases also allow businesses to track sales, inventory, and other important data, which can help them make informed decisions about their operations.

MySQL is one of the most widely used open-source relational database management systems. It is particularly popular because of its ease of use, scalability, and flexibility. MySQL is used by companies of all sizes, from small startups to large enterprises. It is particularly popular for web-based applications, including content management systems, e-commerce platforms, and social networking sites.

One of the key benefits of using MySQL is that it is easy to set up and use. This makes it accessible to businesses that may not have a dedicated IT department or database administrator. MySQL is also scalable, meaning that it can easily accommodate growth as a business expands. It is designed to handle large amounts of data and can be optimized for high-performance applications.

Another important benefit of using MySQL is its flexibility. It can be used for a wide range of applications and can be customized to meet the specific needs of a business. Additionally, MySQL is open-source, meaning that it is constantly being updated and improved by a community of developers. This ensures that the software is always up-to-date and secure.

In conclusion, databases are an essential tool for businesses of all sizes. They allow for efficient data management, which is crucial for making informed business decisions. MySQL is a powerful, flexible, and easy-to-use database management system that is widely used by businesses around the world.

**Connection between DB and Python**

Python is a versatile and widely used programming language that is widely used for various data-centric applications. One of the most important use cases for Python is working with databases, and in particular, relational databases such as MySQL, PostgreSQL, and Oracle. Relational databases are designed to store data in tables with rows and columns, which can be queried, updated, and manipulated in various ways. Python provides various libraries and modules that make it easy to connect to databases, query them, and perform various operations on the data.

To establish a connection between Python and a database, the first step is to choose a suitable Python database library or module. Some of the popular libraries for connecting to databases include MySQLdb, PyMySQL, cx\_Oracle, and psycopg2. Each library has its own advantages and disadvantages, depending on the specific database being used, the Python version, and the use case.

Once a suitable library is chosen, the next step is to establish a connection to the database. This typically involves providing the necessary connection parameters such as the host name, username, password, and database name. Once the connection is established, the Python program can send SQL queries to the database and receive the results.

Python provides a rich set of data structures and objects that can be used to represent the data retrieved from the database. For example, a query that retrieves data from a table can be represented as a list of tuples, where each tuple corresponds to a row in the table. The individual columns in the row can be accessed using indexing or named attributes, depending on the library being used.

Python also provides several frameworks and libraries for building web applications that interact with databases. For example, the Django framework provides a high-level ORM (Object-Relational Mapping) layer that abstracts away much of the low-level details of database connections and queries, allowing developers to focus on the application logic. Other web frameworks such as Flask and Pyramid also provide support for working with databases in Python.

In addition to querying and updating data in databases, Python can also be used for other tasks such as data analysis, machine learning, and data visualization. Python provides a wide range of libraries and tools for these tasks, including pandas, NumPy, SciPy, scikit-learn, TensorFlow, and matplotlib, among others. Many of these libraries can directly read data from databases or can be easily integrated with Python database libraries.

In summary, the connection between Python and databases is crucial for many data-centric applications, and Python provides a rich set of tools and libraries for working with databases of various types. By leveraging these tools and libraries, developers can build powerful and scalable applications that can handle large amounts of data, perform complex analytics, and provide valuable insights to users.

**References**

### Fluent Python: Clear, Concise, and Effective Programming by Luciano Ramalho

ChatGPT

## Database Systems Using Oracle

### [**Learning MySQL: Get a Handle on Your Data**](https://amzn.to/2X7r1sN)

## [Head First Object-Oriented Analysis and Design](https://www.amazon.com/Head-First-Object-Oriented-Analysis-Design/dp/0596008678/?tag=javamysqlanta-20)