ATM Simulator Project: Core Python Concepts

This document summarizes the fundamental Python code elements utilized in the ATM Simulator project, highlighting the procedural structure and data management strategies employed.

1. Data Types and Structures

The project uses only native Python data types to manage the application's state in memory.

Element	Python Data Type	Role in the Project	Example
Account Store	Dictionary (dict)	Stores all static and dynamic account information (username, PIN, balance, transactions) in a single, accessible structure.	ACCOUNT_DATA = { "username": "user123", ... }
Balance / Amount	Float (float)	Used for all currency-related fields to ensure precision, supporting decimal values during deposits and withdrawals.	balance = 5000.00
Credentials / Input	String (str)	Handles all user input (menu choices, username, PIN) and credential storage.	pin = "1234"
Transaction Log	List of Dictionaries	Maintains an **ordered sequence** of all transactions performed during the session for the account statement.	transactions = [{ 'type': 'Deposit', ... }]

2. Control Flow Constructs

Control flow determines the order in which code executes and enforces the rules of the ATM simulation.

Construct	Usage in ATM Simulator	Mechanism
Functions (def)	**Procedural Structure:** The entire application logic is modularized into distinct, reusable functions (sign_in, withdraw_amount, main).	Defines clear separation of concerns, making the code easier to read and debug.
while True Loop	**Main Menu Control:** The core function (main()) uses an infinite loop to ensure the menu is displayed repeatedly until the user explicitly selects the 'Exit' option.	Keeps the application running continuously.
for Loop	**Attempt Limit Enforcement:** Used in the sign_in() function to control security by limiting failed login attempts to three.	for attempt in range(3): ...
Conditional Statements (if, elif, else)	**Business Logic and Validation:** Used extensively for security checks, validating menu inputs, and enforcing financial rules (e.g., checking for sufficient funds before withdrawal).	if amount > balance: # Overdraft check

3. Error Handling

Effective error handling is used to prevent the program from crashing due to unexpected user input.

Construct	Purpose	Contextual Example
try...except ValueError	**Data Integrity:** Catches errors that occur when the user provides non-numeric input (e.g., typing 'ten dollars' instead of '10') when a numerical value (float) is expected for a transaction.	Ensures the program provides a friendly error message instead of crashing.

4. Modules

Minimal use of external modules is a characteristic of this project, relying mainly on one Python standard library.

Module	Usage	Rationale
sys	**System Termination:** Used specifically for the sys.exit() command.	Allows for a clean, controlled shutdown of the program when the user chooses 'Exit' or fails the security attempt limits.