|  |
| --- |
| A Mini Project of ATM MACHINE  Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of  **BACHELOR OF TECHNOLOGY**  COMPUTER SCIENCE AND ENGINEERING  **To**    **Dr. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY**  **LUCKNOW**  **SUBMITTED BY-**  Sonakshi Singh  University Roll No.-2102840100169  Branch-CS  **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  **UNITED INSTITUTE OF TECHNOLOGY, PRAYAGRAJ**  **JANUARY 2023** |

# DECLARATION

I-Sonakshi Singh (2102840100169), students of B. Tech of Computer Science and Engineering hereby declared that I own the full responsibility for the information, results etc provided in this mini project titled **“ATM MACHINE”** submitted to Dr. A.P.J Abdul Kalam Technical University, Lucknow for award of B.Tech (Computer Science and Engineering) degree. I have taken care in all respect to honour the intellectual property right and have acknowledged the contributions of others for using them in this academic purpose. I further declared that in case of any violation of intellectual property right or copyright, I as the candidate would be fully responsible for the same. My supervisor and institute should not be held for full or partial violation of copy right if found at any stage of our degree.

**SONAKSHI SINGH**

**2102840100169**

**BRANCH-CS**

# CERTIFICATE



# ACKNOWLEDGEMENT

It is my proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report . It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost , I wish to record my sincere gratitude to United College Coordinators for their constant support and encouragement in preparation of this report and for making available various facilities needed to prepare this report.

Last but not the least , I wish to thank my parents for financing our studies in this college as well as for constantly encouraging me to learn different things in engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledgement.

**SONAKSHI SINGH**

# 

# ABSTRACT

Simple **ATM System** project is written in Python. The project file contains a python script (simpleatm.py). This is a simple

console based system which is very easy to use. Talking about the system, it contains limited functions which only includes

Withdrawing and Depositing amount. Here, at first the user has to enter existing account holder name, when the name matches the

system proceed toward the next procedure i.e asking pin number.

When a user passes all these sign-in procedures, he/she can use the features. It is too easy to use, he/she just has to enter the

amount whether in deposit section or the withdraw section. The system calculates the total remaining balance of the respective account and displays to the user.

This simple console based **ATM system** provides the simple account balance management of a respective account. It contains fewer features but the essential one. There is no database

connection or neither any external text or other files used in this

mini project to save user’s data. After each transaction using the

system, the system redirects the user to the login page and he/she can log in again with the other account too.

In order to run the project, you must have installed Python, on your PC. This is a simple Console Based system, specially

written for the beginners. Simple ATM System in Python project with source code is free to download. Use for education purpose only! For the project demo, have a look at the image slider below.

**Features:**

Sign In

Withdraw Amount

Deposit Amount

# TABLE OF CONTENTS

* DECLARATION
* CERTIFICATE
* ACKNOWLEDMENT
* ABSTRACT
* TABLE OF CONTENT
* INTRODUCTION TO PYTHON
* HISTORY OF PYTHON
* UNDESTANDING OPERATORS
* VARIABLES AND DATA TYPES
* CONDITIONAL STATEMENTS
* LOOPING CONSTRUCTS
* FUNCTIONS
* DATA STRUCTURE
* LISTS
* DICTIONARIES
* PYTHON FEATURES
* PROJECT CODE
* OUTPUT
* CONCLUSION
* REFERENCE

## Introduction to Python

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

**Below are some facts about Python Programming Language:**

* Python is currently the most widely used multi-purpose, high-level programming language.
* Python allows programming in Object-Oriented and Procedural paradigms.
* Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
* Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.
* Thebiggeststrength of Python is hugecollection of standard librarywhich can be used for the following:
* Machine Learning
* GUI Applications (like Kivy, Tkinter, PyQt etc. )
* Web frameworks like Django (used by YouTube, Instagram, Dropbox)
* Image processing (like OpenCV, Pillow)
* Web scraping (like Scrapy, BeautifulSoup, Selenium)
* Test frameworks
* Multimedia

## History of Python

* Python was developed by **Guido van Rossum in 1991** at the National Research Institute for Mathematics and Computer Science in the Netherlands.

* Python is derived from many other languages, including **ABC, Modula-3, C, C++, Algol-68, Small Talk, and Unix shell and other scripting languages.**

* Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

* Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

### Understanding Operators

1. **Arithmetic operators:**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

|  |  |  |
| --- | --- | --- |
| OPERATOR | DESCRIPTION | SYNTAX |
| + | Addition: adds two operands | x+y |
| - | Subtraction: subtracts two operands | x-y |
| \* | Multiplication: multiplies two operands | x\*y |
| / | Division(float): divide the first operand by the second | x/y |
| // | Division(floor): divides the first operand by the second | x//y |
| % | Modulus: return the remainder when first operand is divided by the second | x%y |
| \*\* | Power : Returns first raised to power second | x\*\*y |

1. **Relational Operators:**

Relational operators compares the values. It either returns **True** or **False** according to the condition.

|  |  |  |
| --- | --- | --- |
| OPERATOR | DESCRIPTION | SYNTAX |
| > | Greater than: True if left operand is greater than the right | x > y |
| < | Less than: True if left operand is less than the right | x < y |
| = = | Equal to: True if both operands are equal | x = = y |
| ! = | Not equal to - True if operands are not equal | x ! = y |
| > = | Greater than or equal to: True if left operand is greater than or equal to the right | x > = y |
| < = | Less than or equal to: True if left operand is less than or equal to the right | x < = y |

1. **Logical operators:**

Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

|  |  |  |
| --- | --- | --- |
| OPERATOR | DESCRIPTION | SYNTAX |
| and | Logical AND: True if both the operands are true | x and y |
| or | Logical OR: True if either of the operands is true | x or y |
| not | Logical NOT: True if operand is false | not x |

1. **Bitwise operators:**

Bitwise operators acts on bits and performs bit by bit operation.

|  |  |  |
| --- | --- | --- |
| OPERATOR | DESCRIPTION | SYNTAX |
| & | Bitwise AND | x & y |
| | | Bitwise OR | x | y |
| ~ | Bitwise NOT | ~ x |
| ^ | Bitwise XOR | x ^ y |
| >> | Bitwise RIGHT SHIFT | x >> |
| << | Bitwise LEFT SHIFT | x << |

1. **Assignment operators:**

Assignment operators are used to assign values to the variables.

|  |  |  |
| --- | --- | --- |
| OPERATOR | DESCRIPTION | SYNTAX |
| = | Assign value of right side of expression to left side operand | x=a + b |
| + = | Add AND: Add right side operand with left side operand and then assign to left operand | x+=a |
| - = | Subtract AND: Subtract right operand from left operand and then assign to left operand | x-=a |
| \* = | Multiply AND: Multiply right operand with left operand and then assign to left operand | x\*=a |
| / = | Divide AND: Divide left operand with right operand and then assign to left operand | x/=a |
| % = | Modulus AND: Takes modulus using left and right operands and assign result to left operand | x%=a |

1. **Special operators:**

There are some special type of operators like-

1. **Identity operators:**

**Is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

**is** True if the operands are identical

**is not** True if the operands are not identical

1. **Membership operators**:

**in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

**in** True if value is found in the sequence

**not in** True if value is not found in the sequence

### Variables and Data Types

**Variables:**

1. **Python Variables Naming Rules:**

There are certain rules to what you can name a variable (called an identifier).

* + Python variables can only begin with a letter(A-Z/a-z) or an underscore ( \_ ).
  + The rest of the identifier may contain letters(A-Z/a-z), underscores ( \_ ), and numbers (0-9).
  + Python is case-sensitive, and so are Python identifiers. Name and name are two different identifiers.

1. **Assigning and Reassigning Python Variables:**

* + To assign a value to Python variables, you don’t need to declare its type.
  + You name it according to the rules stated in section 2a, and type the value after the equal sign (=).
  + You can’t put the identifier on the right-hand side of the equal sign.
  + Neither can you assign Python variables to a keyword.

1. **Multiple Assignment:**

* + You can assign values to multiple Python variables in one statement.
  + You can assign the same value to multiple Python variables.

1. **Deleting Variables:**

* + You can also delete Python variables using the keyword ‘del’.

**Data Types:**

1. **Python Numbers:**

There are four numeric Python data types.

* 1. **int:** int stands for integer. This Python Data Type holds signed integers. We can use the type() function to find

which class it belongs to.

* 1. **float:** This Python Data Type holds floating-point real values. An int can only store the number 3, but float can store 3.25 if you want.
  2. **long:** This Python Data type holds a long integer of unlimited length. But this construct does not exist in Python 3.x.
  3. **complex:** This Python Data type holds a complex number. A complex number looks like this: a+bj Here, a and b are the real parts of the number, and j is imaginary.

1. **Strings:**

A string is a sequence of characters. Python does not have a char data type, unlike C++ or Java. You can delimit a string using single quotes or double-quotes.

* 1. **Spanning a String Across Lines:** To span a string across multiple lines, you can use triple quotes.
  2. **Displaying Part of a String:** You can display a character from a string using its index in the string. Remember, indexing starts with 0.
  3. **String Formatters:** String formatters allow us to print characters and values at once. You can use the % operator.
  4. **String Concatenation:** You can concatenate(join) strings using + operator. However, you cannot concatenate values of different types.

1. **Python Lists:**

A list is a collection of values. Remember, it may contain different types of values.

To define a list, you must put values separated with commas in square brackets. You don’t need to declare a type for a list either.

* 1. **Slicing a List:** You can slice a list the way you’d slice a string- with the slicing operator. Indexing for a list begins with 0, like for a string. A Python doesn’t have arrays.
  2. **Length of a List:** Python supports an inbuilt function to calculate the length of a list.
  3. **Reassigning Elements of a List:** A list is mutable. This means that you can reassign elements later on.
  4. **Iterating on the List:** To iterate over the list we can use the for loop. By iterating, we can access each element one by one which is very helpful when we need to perform some operations on each element of list.
  5. **Multidimensional Lists:** A list may have more than one dimension. Have a detailed look on this in Data Flair’s tutorial on Python Lists.

1. **Python Tuples:**

A tuple is like a list. You declare it using parentheses instead.

1. **Accessing and Slicing a Tuple**: You access a tuple the same way as you’d access a list. The same goes for slicing it.
2. **A tuple is Immutable**: Python tuple is immutable. Once declared, you can’t change its size or elements.

**E. Dictionaries:**

A dictionary holds key-value pairs. Declare it in curly braces, with pairs separated by commas. Separate keys and values by a colon (:). The type() function works with dictionaries too.

1. **Accessing a Value**: To access a value, you mention the key in square brackets.
2. **Reassigning Elements**: You can reassign a value to a key.
3. **List of Keys**: Use the keys() function to get a list of keys in the dictionary.

1. **Bool:** 
   1. Boolean value can be True or False.

1. **Set:** 
   1. set can have a list of values. Define it using curly braces. It returns only one instance of any value present more than once. However, a set is unordered, so it doesn’t support indexing. Also, it is mutable. You can change its elements or add more. Use the add() and remove() methods to do so.

1. **Type Conversion:**

Since Python is dynamically-typed, you may want to convert a value into another type. Python supports a list of functions for the same.

1. **int()**

1. **float()**

1. **bool()**

1. **set()**

1. **list()**

1. **tuple()**

1. **str()**

### Conditional Statements

**a. If statements:**

If statement is one of the most commonly used conditional statement in most of the programming languages. It decides whether certain statements need to be executed or not. If statement checks for a given condition, if the condition is true, then the set of code present inside the if block will be executed. The If condition evaluates a Boolean expression and executes the block of code only when the Boolean expression becomes TRUE.

**Syntax:**

if (Boolean expression):

Block of code #Set of statement to execute if the condition is true **b. If-else statements:**

The statement itself tells that if a given condition is true then execute the statements present inside if block and if the condition is false then execute the else block. Else block will execute only when the condition becomes false, this is the block where you will perform some actions when the condition is not true. If-else statement evaluates the Boolean expression and executes the block of code present inside the if block if the condition becomes TRUE and executes a block of code present in the else block if the condition becomes FALSE.

**Syntax:**

if(Boolean expression):

Block of code #Set of statements to execute if condition is true

else:

Block of code #Set of statements to execute if condition is false **c. elif statements:**

In python, we have one more conditional statement called elif statements. elif statement is used to check multiple conditions only if the given if condition false. It’s similar to an if-else statement and the only difference is that in else we will not check the condition but in elif we will do check the condition. Elif statements are similar to if-else statements but elif statements evaluate multiple conditions.

**Syntax:** if (condition):

#Set of statement to execute if condition is true elif (condition):

#Set of statements to be executed when if condition is false and elif condition is true

else:

#Set of statement to be executed when both if and elif conditions are false

**d. Nested if-else statements:**

Nested if-else statements mean that an if statement or if-else statement is present inside another if or if-else block. Python provides this feature as well, this in turn will help us to check multiple conditions in a given program. An if statement present inside another if statement which is present inside another if statements and so on.

**Nested if Syntax:**

if(condition):

#Statements to execute if condition is true if(condition):

#Statements to execute if condition is true

#end of nested if

#end of if

**Nested if-else Syntax:**

if(condition):

#Statements to execute if condition is true if(condition):

#Statements to execute if condition is true else:

#Statements to execute if condition is false else:

#Statements to execute if condition is false

**e. elif Ladder:**

We have seen about the elif statements but what is this elif ladder. As the name itself suggests a program which contains ladder of elif statements or elif statements which are structured in the form of a ladder. This statement is used to test multiple expressions.

**Syntax:** if (condition):

#Set of statement to execute if condition is true elif (condition):

#Set of statements to be executed when if condition is false and elif condition is true

elif (condition):

#Set of statements to be executed when both if and first elif condition is false and second elif condition is true elif (condition):

#Set of statements to be executed when if, first elif and second elif conditions are false and third elif statement is true

else:

#Set of statement to be executed when all if and elif conditions are false

### Looping Constructs

**Loops:**

1. **while loop:**

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

**Syntax:**

while expression:

statement(s)

1. **for loop:**

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

**Syntax:**

for iterating\_var in sequence:

statements(s)

1. **nested loops:**

You can use one or more loop inside any another while, for or do...while loop.

**Syntax of nested for loop:**

for iterating\_var in sequence:

for iterating\_var in sequence:

statements(s)

statements(s)

**Syntax of nested while loop:**

while expression:

while expression:

statement(s) statement(s)

**Loop Control Statements:**

1. **break statement:**

Terminates the loop statement and transfers execution to the statement immediately following the loop.

1. **continue statement:**

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

1. **pass statement:**

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

### Functions

1. **Built-in Functions or pre-defined functions:**

These are the functions which are already defined by Python. For example: id (), type (), print (), etc.

1. **User-Defined Functions:**

These are functions that are defined by the users for simplicity and to avoid repetition of code. It is done by using def function.

### Data Structure

Python has implicit support for Data Structures which enable you to store and access data. These structures are called List, Dictionary, Tuple and Set.

**Lists:**

Lists in Python are the most versatile data structure. They are used to store heterogeneous data items, from integers to strings or even another list! They are also mutable, which means that their elements can be changed even after the list is created.

**Creating Lists**

Lists are created by enclosing elements within [square] brackets and each item is separated by a comma.

**Creating lists in Python**

Since each element in a list has its own distinct position, having duplicate values in a list is not a problem.

**Accessing List elements**

To access elements of a list, we use Indexing. Each element in a list has an index related to it depending on its position in the list. The first element of the list has the index 0, the next element has index 1, and so on. The last element of the list has an index of one less than the length of the list.

**Indexing in Python lists**

While positive indexes return elements from the start of the list, negative indexes return values from the end of the list. This saves us from the trivial calculation which we would have to otherwise perform if we wanted to return the nth element from the end of the list. So instead of trying to return

List\_name[len(List\_name)-1]element, we can simply write List\_name[-1]. Using negative indexes, we can return the nth element from the end of the list easily. If we wanted to return the first element from the end, or the last index, the associated index is -1. Similarly, the index for the second last element will be -2, and so on. Remember, the 0th index will still refer to the very first element in the list.

**Appending values in Lists**

We can add new elements to an existing list using the append() or insert() methods: append() – Adds an element to the end of the list

insert() – Adds an element to a specific position in the list which needs to be specified along with the value

**Removing elements from Lists**

Removing elements from a list is as easy as adding them and can be done using the remove() or pop() methods: remove() – Removes the first occurrence from the list that matches the given value

pop() – This is used when we want to remove an element at a specified index from the list. However, if we don’t provide an index value, the last element will be removed from the list.

**Sorting Lists**

On comparing two strings, we just compare the integer values of each character from the beginning. If we encounter the same characters in both the strings, we just compare the next character until we find two differing characters.

**Concatenating Lists**

We can even concatenate two or more lists by simply using the + symbol. This will return a new list containing elements from both the lists:

### Dictionaries

Dictionary is another Python data structure to store heterogeneous objects that are immutable but unordered.

**Generating Dictionary**

* Dictionaries are generated by writing keys and values within a **{**curly**}**bracket separated by a semi-colon. And each key-value pair is separated by a comma :
* Using the key of the item, we can easily extract the associated value of the item
* Dictionaries are very useful to access items quickly because, unlike lists and tuples, a dictionary does not have to iterate over all the items finding a value.
* Dictionary uses the item key to quickly find the item value. This concept is called **Hashing**.

**Accessing keys and values**

You can access the keys from a dictionary using the **keys()** method and the values using the **values()** method. These we can view using a *for-loop* or turn them into a list using **list()**

## PYTHON FEATURES

Python's features include:

* **Easy-to-learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* **Easy-to-read**: Python code is more clearly defined and visible to the eyes.
* **Easy-to-maintain**: Python's source code is fairly easy-to-maintain.
* **A broad standard library**: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* **Databases**: Python provides interfaces to all major commercial databases.
* **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* **Scalable**: Python provides a better structure and support for large programs than shell scripting. Apart from the above-mentioned features, Python has a big list of good features, few are listed below −
* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte- code for building large applications.
* It provides very high-level dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.

### PROJECT CODE

from optparse import Option

from pickle import TRUE

import time print()

print(" !! WELCOME TO MY ATM MACHINE !!") print() print(" NAME :SONAKSHI SINGH") print(" ROLL NO: 2102840100169") print(" BRANCH :COMPUTER SCIENCE")

print(" SECTION: D [CS]")

print("Please Insert Your Card")

print()

time.sleep(1)

password = 1234

pin1 = 4567

pin=int(input("Enter Your Pin: "))

print()

balance=10000000

if (pin == password or pin==pin1): while TRUE:

print(""" PRESS :- 1: BALANCE

2: CASH WITHDRAW

3: CASH DEPOSIT

4: EXIT

""")

try:

Option=int(input("PLEASE ENTER YOUR CHOICE: ")) except:

print("PLEASE ENTER VALID OPTION")

if Option==1:

print()

print(f">>> YOUR CURRENT BALANCE IS: {balance}") print()

if Option==2:

withdraw\_amount=int(input("PLEASE ENTER WITHDRAW AMOUNT: ")) """if(withdraw\_amount <=balance):

print(f"UNSUFFICIENT BALANCE IN YOUR ACCOUNT") print()

else:"""

balance=balance-withdraw\_amount print()

print(f">>> {withdraw\_amount} IS DEBITED FROM YOUR ACCOUNT") print(f">>> YOUR UPDATED BALANCE IS: {balance}") print()

if Option==3 :

print()

deposit\_amount=int(input("PLEASE ENTER DEPOSIT AMOUNT: ")) balance=balance+ deposit\_amount print()

print(f">>> {deposit\_amount} IS CREDITED TO YOUR ACCOUNT") print(f">>> YOUR UPDATED BALANCE IS: {balance}") print()

if Option==4: print(" Thank You!! ") break else:

print("WRONG PIN !! PLEASE RE-INSERT YOUR CARD...")

### CONCLUSION

Practical knowledge means the visualization of the knowledge, which we read in our books. For this, we perform experiments and get observations. Practical knowledge is very important in every field. One must be familiar with the problems related to that field so that he may solve them and become a successful person. After achieving the proper goal in life, an engineer has to enter in professional life. According to this life, he has to serve an industry, may be public or private sector or self-own. For the efficient work in the field, he must be well aware of the practical knowledge as well as theoretical knowledge.

Due to all above reasons and to bridge the gap between theory and practical, our engineering curriculum provides a practical training of 60 Hours. During this period a student worked in the industry and get well all type of experience and knowledge about the working of companies and hardware and software tools.

I have undergone my 60 Hours summer training in 3rd semester ,This report is based on the knowledge, which I acquired during my 60 Hours of summer training.

**REFERENCES**

**- Python programming handbook**

### - en.wikipedia.org

**-** [**WWW.python.orgs**](http://www.python.orgs/)