**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Particulars** | **Page No.** |
| 1 | Introduction |  |
| 2 | Technology Used |  |
| 3 | What can Python do? |  |
| 4 | Why Python? |  |
| 5 | Hardware and Software used |  |
| 6 | Aim |  |
| 7 | Program Code |  |
| 8 | Output |  |
| 9 | Flowchart |  |
| 10 | Bibliography |  |

**INTRODUCTION**

This project is a terminal-based net-banking app, in which users can :

1. Create and log into a password protected account
2. Deposit money
3. Make transactions with other users
4. Create fixed deposits and gain interest

All user-data is stored in and retrieved from a MySQL database. For the application itself, Python is used.

One of the main features of this project in terms of development is its state machine architecture, which is discussed in detail later in this document.

**WHY PYTHON?**

* **Cross-platform Language**: Python can run equally well on variety of platforms – Windows, Linus/UNIX, smartphones, etc.
* **Simple and expressive syntax:** Python has a simple syntax similar to the English language. It is thus very expressive with fewer lines of code and simplicity compared to other popular languages like C++, Java etc.
* **Quick prototyping:** Python runs on an interpreter system, so the code can be executed as soon as it is written. This, along with its simplicity, means that prototyping can be very quick.
* **Multi-paradigm:** Python can be written in a procedural way, an object-oriented way or a functional way.

**SYSTEM SPECS**

|  |  |
| --- | --- |
| **Operating System** | Windows 10 |
| **Processor** | Intel Core i3 7th gen @ 2.30 Ghz |
| **RAM** | 12 GB |
| **Hard disk** | SSD 233 GB, HDD 932 GB |

**AIM**

To create a net-banking client application with terminal-based UI.

**Some Background**

As mentioned before, the state machine architecture of this application is a highlight of this project.

Based on what functionality the user wants to access, different kinds of processing have to be done. The idea of a state arises from this situation naturally. Based on user input, we will set a certain “state”, and based on the current state, some processing will be done. Each state can also change the current state to a different one, allowing navigation between different states.

A naïve implementation would declare constants that represent different states, and would check in an if-elif chain what state is currently set, and run code based on that, like this :

1 | STATE0 = 0

2 | STATE1 = 1

3 | currentState = 0

4 | ​

5 | while True: *# main process-loop*

6 |  if currentState == STATE0:

7 |  *# STATE0'S processing*

8 |

9 |  inp = userInput()

10|

11|  if inp == "change\_state":

12|             *# some user-input condition*

13|  currentState = STATE1

14|

15|  continue

16|

17|  elif currentState == STATE1:

18|  *# STATE1'S processing*

19|

20|  inp = userInput()

21|

22|  if inp == "change\_state":

23|  currentState = STATE0

24|

25|  continue

However, this if-elif chain can quickly grow very large. Since the states are being set by the code itself explicitly, there shouldn’t be any need to check for the state in each process-loop. Moreover, the implementation for different states cannot be separated and thus modularization cannot be achieved, which would be desirable from a code-design standpoint.

The problem is that the state in the above code is represented by an integer object, which does not contain any information about what kind of processing it needs. Thus, the current state needs to be checked and its implementation has to be provided by the main process-loop itself. However, if the state was represented by an object that itself contained information about the required processing, then we could just use that information without caring what the current state exactly is. This state-object can be a class that contains a process() function, which is called by the main process-loop. This eliminates the need of if-checks altogether. Also, since the class definitions can be written separately, it improves code-design by allowing modularization. This design is exemplified by the application code following this page.

**PROGRAM**

**CODE**

**SAMPLE**

**OUTPUT**

**BIBLIOGRAPHY**

* Sumita Arora Textbook Computer Science with Python Class XII