**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**

0 | import time

1 | from getpass import getpass

2 | import datetime as dt

3 | from dateutil.relativedelta import relativedelta

4 | import mysql.connector as sqlconn

5 | from mysql.connector import DataError, DatabaseError, OperationalError, l NotSupportedError, IntegrityError, ProgrammingError, InternalError

6 | ​

7 | try:

8 | ​

9 |     currentState = None

10|     TIMEDELTA = 2

11|     currentDate = None

12| ​

13|     db = sqlconn.connect(host="localhost", user="root", l l l l ll l password="VSE@2022", database="bank", charset="utf8")

14|     crsr = db.cursor(buffered=True)

15|

16|     def EXIT(code=0):

17|         db.close()

18|         exit(code)

19| ​

20|     def execute(query : str, args : tuple) -> None:

21|         crsr.execute(query.format(\*args))

22| ​

23|     def resultExists(result):

24|         if len(result):

25|             return True

26|         else:

27|             return False

28| ​

29|     def getBalance(username : str) -> int:

30|         Q\_GET\_BALANCE = ("SELECT balance "

31|                          "FROM account "

32|                          "WHERE username = '{}';")

33| ​

34|         execute(Q\_GET\_BALANCE, (username,))

35|         return crsr.fetchone()[0]

36| ​

37|     def c\_changeBalance(username : str, change : int) -> None:

38|         QC\_CHANGE\_BALANCE = ("UPDATE Account "

39|                              "SET balance = balance + {1} "

40|                              "WHERE username = '{0}'; ")

41|

42|         execute(QC\_CHANGE\_BALANCE, (username, change))

43| ​

44|     def userExists(username : str) -> bool:

45|         Q\_CHECK\_USERNAME = ("SELECT username "

46|                             "FROM Users "

47|                             "WHERE username = '{}';")

48| ​

49|         execute(Q\_CHECK\_USERNAME, (username,))

50| ​

51|         if len(crsr.fetchall()) != 0:

52|             return True

53|         else:

54|             return False

55| ​

56|     def checkFDExists(username : str, fdName : str) -> bool:

57|         Q\_CHECK\_FD\_EXISTS = ("SELECT \* FROM FixedDepo "

58|                              "WHERE username = '{}' AND fdName = '{}'; l ")

59| ​

60|         execute(Q\_CHECK\_FD\_EXISTS, (username, fdName))

61| ​

62|         if len(crsr.fetchall()) != 0:

63|             return True

64|         else:

65|             return False

66| ​

67|     def intInput(prompt : str, failMsg : str = "Invalid input.") -> l l int:

68|         while True:

69|             inpStr = input(prompt).strip()

70| ​

71|             if not inpStr.isdigit():

72|                 print(failMsg)

73|             else:

74|                 return int(inpStr)

75|

76|     def getUpdates(username, date=None):

77|         \_Q\_GET\_UPDATES\_ALL = ("SELECT baseContent, extraContent, | updateDate "

78|                               "FROM Updates "

79|                               "WHERE username = '{}';")

80| ​

81|         \_Q\_GET\_UPDATES\_DAY = ("SELECT baseContent, extraContent,

updateDate "

82|                               "FROM Updates "

83|                               "WHERE username = '{}' "

84|                               "AND updateDate = '{}'")

85| ​

86|         if date:

87|             execute(\_Q\_GET\_UPDATES\_DAY, (username, date))

88|             return crsr.fetchall()

89|         else:

90|             execute(\_Q\_GET\_UPDATES\_ALL, (username, ))

91|             return crsr.fetchall()

92| ​

93|     def c\_createUpdate(username, baseContent, extraContent="No

comment", \_date=None):

94|         \_QC\_CREATE\_UPDATE = ("INSERT INTO Updates "

95|                              "VALUES "

96|                              "('{}', '{}', '{}', '{}')")

97| ​

98|         if \_date:

99|             execute(\_QC\_CREATE\_UPDATE, (username, baseContent,

extraContent, \_date))

100|         else:

101|             execute(\_QC\_CREATE\_UPDATE, (username, baseContent,

extraContent, currentDate))

102| ​

103|     def getUserInfo(username):

104|         \_Q\_GET\_USER = ("SELECT firstname, lastname, age, phone,

inactive "

105|                        "FROM Users "

106|                        "WHERE username = '{}' ;")

107| ​

108|         execute(\_Q\_GET\_USER, (username,))

109|         return crsr.fetchone()

110| ​

111|     class LockedState:

112|         def \_\_init\_\_(self):

113|             pass

114| ​

115|         def process(self):

116|             global currentState

117| ​

118|             print("===================================================

===================")

119|             print("Enter username and password to view details or

create a new account")

120|             print("(1) Login")

121|             print("(2) Create an account")

122|             print("(3) Quit")

123|             print()

124| ​

125|             option = intInput("(Option) -> ")

126| ​

127|             if option == 1:

128|                 currentState = LoginState()

129| ​

130|             elif option == 2:

131|                 currentState = CreateAccountState()

132| ​

133|             elif option == 3:

134|                 EXIT()

135| ​

136|             else:

137|                 print()

138|                 print("Please choose a valid option.")

139| ​

140|     class LoginState:

141|         \_Q\_LOGIN\_USER = ("SELECT username, password "

142|                          "FROM Users "

143|                          "WHERE username = '{}'; ")

144| ​

145|         def \_\_init\_\_(self):

146|             pass

147| ​

148|         def \_login(self, username : str, password : str) -> int:

149|             global currentState

150| ​

151|             execute(self.\_Q\_LOGIN\_USER, (username,))

152|             record = crsr.fetchone()

153| ​

154|             if record == None:

155|                 print("Username not found.")

156|                 currentState = LockedState()

157|                 return

158|

159|             if record[1] != password:

160|                 print("Incorrect password.")

161|                 currentState = LockedState()

162|                 return

163| ​

164|             print("Logged in successfully.")

165| ​

166|             currentState = UnlockedState(username)

167| ​

168|         def process(self):

169|             print("=======================================")

170|             username = input("(Enter Username) -> ").strip()

171|             password = getpass("(Enter Password) -> ").strip()

172|             print()

173| ​

174|             self.\_login(username, password)

175| ​

176|     class CreateAccountState:

177|         \_QC\_CREATE\_USER = ("INSERT INTO Users VALUES "

178|                            "('{}', '{}', '{}', '{}', {}, '{}', {}); ")

179| ​

180|         \_QC\_CREATE\_ACCOUNT = ("INSERT INTO account "

181|                               "VALUES "

182|                               "({}, '{}', {}, '{}'); ")

183| ​

184|         def \_\_init\_\_(self):

185|             pass

186|

187|         def \_createNewUser(self, username : str, password : str,

firstname : str,

188|                          lastname : str, age : int, phone : int) ->

int:

189|             execute(self.\_QC\_CREATE\_USER, (password, username,

firstname, lastname, age, phone, 0))

190|             execute(self.\_QC\_CREATE\_ACCOUNT, (0, str(currentDate), 0,

username))

191|             db.commit()

192| ​

193|         def process(self):

194|             global currentState

195| ​

196|             print("========================================")

197|             print("(0) Create account")

198|             print("(1) Abort")

199|             print()

200| ​

201|             option = intInput("(Option) -> ")

202| ​

203|             if option == 0:

204|                 print()

205|                 username = input("(Enter NEW Username) -> ").strip()

206| ​

207|                 if userExists(username):

208|                     print()

209|                     print("Username not unique.")

210|                     return

211| ​

212|                 while True:

213|                     password = input("(Enter NEW Password) -> ").strip()

214|                     confirmPassword = getpass("(Enter password for confirmation) -> ").strip()

215| ​

216|                     if password == confirmPassword:

217|                         break

218| ​

219|                     print("Passwords do not match. Enter again.")

220| ​

221|                 firstname = input("(Enter first name) -> ").strip()

222|                 lastname = input("(Enter last name) -> ").strip()

223|                 age = intInput("(Enter age) -> ")

224|                 phone = intInput("(Enter phone no.) -> ")

225|                 print()

226| ​

227|                 self.\_createNewUser(username, password, firstname, lastname, age, phone)

228| ​

229|                 currentState = UnlockedState(username)

230| ​

231|             elif option == 1:

232|                 currentState = LockedState()

233| ​

234|             else:

235|                 print()

236|                 print("Please choose a valid option")

237| ​

238|     class UnlockedState:

239|         def \_\_init\_\_(self, username : str):

240|             self.\_username = username

241| ​

242|         def process(self):

243|             global currentState

244|             global currentDate

245| ​

246|             # print and remove updates

247|             balance = getBalance(self.\_username)

248|             updates = getUpdates(self.\_username, currentDate)

249| ​

250|             print("===================================")

251|             print(currentDate)

252|             print(f"BALANCE: {balance}")

253|             if resultExists(updates):

254|                 print("TODAY'S UPDATES:", end=" ")

255|                 for content, \_, \_\_ in updates:

256|                     print(f"{content}", end=", ")

257|             print()

258|             print("(0) Logout")

259|             print("(1) Pay")

260|             print("(2) Deposit")

261|             print("(3) Create a fixed deposit")

262|             print("(4) Modify/View fixed deposits")

263|             print("(5) View all updates for your account")

264|             print()

265| ​

266|             option = intInput("(Option) -> ")

267| ​

268|             if option == 1:

269|                 currentState = PayState(self.\_username)

270| ​

271|             elif option == 2:

272|                 currentState = DepositState(self.\_username)

273| ​

274|             elif option == 3:

275|                 currentState = CreateFDState(self.\_username)

276| ​

277|             elif option == 0:

278|                 currentState = LockedState()

279| ​

280|             elif option == 4:

281|                 currentState = ViewFDState(self.\_username)

282|

283|             elif option == 5:

284|                 currentState = ViewUpdatesState(self.\_username)

285| ​

286|             else:

287|                 print()

288|                 print("Please choose a valid option.")

289| ​

290|     class PayState:

291|         \_QC\_PAY\_USER = ("INSERT INTO transactions "

292|                         "(payerID, receiverID, transDate, amount, comment) "

293|                         "VALUES "

294|                         "('{}', '{}', '{}', {}, '{}'); ")

295|

296|         \_Q\_GETUSERPASSWORD = ("SELECT password "

297|                               "FROM Users "

298|                               "WHERE username = '{}'; ")

299| ​

300|         def \_\_init\_\_(self, username : str):

301|             self.\_username = username

302| ​

303|         def \_pay(self, receiverName : str, amount : float, comment: str) -> int:

304|             global currentState

305| ​

306|             global currentState

307|             balance = getBalance(self.\_username)

308| ​

309|             if receiverName == self.\_username:

310|                 print("You cannot pay yourself.")

311|                 return

312| ​

313|             if not userExists(receiverName):

314|                 print("This receiver does not exist.")

315|                 return

316| ​

317|             if amount == 0:

318|                 print("Enter a valid amount to pay.")

319|                 return

320| ​

321|             if amount > balance:

322|                 print("You do not have sufficient balance.")

323|                 return

324| ​

325|             inpPwd = getpass("(Enter password to proceed with payment) -> ")

326|             execute(self.\_Q\_GETUSERPASSWORD, (self.\_username, ))

327|             userPwd = crsr.fetchone()[0]

328| ​

329|             if inpPwd != userPwd:

330|                 print("Incorrect password, aborting payment.")

331|                 return

332| ​

333|             c\_changeBalance(self.\_username, -amount)

334|             execute(self.\_QC\_PAY\_USER, (self.\_username, receiverName, str(currentDate), amount, comment))

335|             c\_changeBalance(receiverName, amount)

336| ​

337|             recFirstName = getUserInfo(receiverName)[0]

338|             userFirstName = getUserInfo(self.\_username)[0]

339|             c\_createUpdate(receiverName, f"{userFirstName} paid {amount}", f"{comment}")

340|             c\_createUpdate(self.\_username, f"Paid {amount} to {recFirstName}", f"{comment}")

341| ​

342|             db.commit()

343| ​

344|             print("Transaction made successfully.")

345| ​

346|         def process(self):

347|             global currentState

348| ​

349|             print("===========================")

350|             print("(0) Pay to another user")

351|             print("(1) Abort")

352|             print()

353| ​

354|             option = intInput("(Option) -> ")

355| ​

356|             if option == 0:

357|                 print()

358|                 receiverName = input("(Enter username of receiver) -> ").strip()

359|                 amount = intInput("(Enter amount to pay) -> ")

360|                 comment =  input("Enter comment (optional)) -> ").strip()

361|                 print()

362| ​

363|                 if not comment:

364|                     comment = "No comment"

365| ​

366|                 self.\_pay(receiverName, amount, comment)

367| ​

368|             elif option == 1:

369|                 currentState = UnlockedState(self.\_username)

370| ​

371|             else:

372|                 print()

373|                 print("Please choose a valid option.")

374| ​

375|     class DepositState:

376|         def \_\_init\_\_(self, username : str):

377|             self.\_username = username

378| ​

379|         def \_deposit(self, amount : int) -> None:

380|             c\_changeBalance(self.\_username, amount)

381|             c\_createUpdate(self.\_username, f"Deposit {amount}")

382|             db.commit()

383| ​

384|         def process(self):

385|             global currentState

386| ​

387|             print("======================================================================")

388|             amount = intInput("(Enter amount to deposit (cash to digital money)) -> ")

389|             self.\_deposit(amount)

390| ​

391|             currentState = UnlockedState(self.\_username)

392| ​

393|     class CreateFDState:

394|         \_QC\_CREATE\_FD = ("INSERT INTO FixedDepo "

395|                          "(fdName, username, principal, interest, creationdate, timeperiod, maturedate) "

396|                          "VALUES('{}', '{}', {}, {}, '{}', {}, '{}'); ")

397| ​

398|         def \_\_init\_\_(self, username : str):

399|             self.\_username = username

400| ​

401|         def \_createFD(self, name : str, amount : int, period : int) -> None:

402|             if checkFDExists(self.\_username, name):

403|                 print("FD with this name already exists")

404|                 return

405| ​

406|             if getBalance(self.\_username) < amount:

407|                 print("You do not have sufficient balance.")

408|                 return

409| ​

410|             c\_changeBalance(self.\_username, -amount)

411|             execute(self.\_QC\_CREATE\_FD, (name, self.\_username, amount, 2, str(currentDate), period,

412|                     currentDate + relativedelta(years=period)))

413|             c\_createUpdate(self.\_username, f"Create {name} FD")

414|             db.commit()

415|             print("FD created successfully.")

416| ​

417|         def process(self):

418|             global currentState

419| ​

420|             print("======================")

421|             print("(0) Create new FD")

422|             print("(1) Return")

423|             print()

424| ​

425|             option = intInput("(Option) -> ")

426| ​

427|             if option == 0:

428|                 print()

429|                 name = input("(Enter FD name) -> ")

430|                 amount = intInput("(Enter amount) -> ")

431|                 period = intInput("(Enter time period in years (under 10)) -> ")

432|                 print()

433| ​

434|                 self.\_createFD(name, amount, period)

435| ​

436|             elif option == 1:

437|                 currentState = UnlockedState(self.\_username)

438| ​

439|             else:

440|                 print()

441|                 print("Please choose a valid option.")

442| ​

443|     class ViewFDState:

444|         \_Q\_GET\_FD\_DETAILS = ("SELECT \* FROM FixedDepo "

445|                              "WHERE username = '{}' AND fdName = '{}'; ")

446|         \_QC\_WITHDRAW\_FD = ("UPDATE FixedDepo "

447|                            "SET withdrawn = 1 "

448|                            "WHERE username = '{}' AND fdName = '{}'; ")

449|         \_Q\_GET\_ALL\_FDS = ("SELECT fdName FROM FixedDepo "

450|                           "WHERE username = '{}'; ")

451| ​

452|         def \_\_init\_\_(self, username : str):

453|             self.\_username = username

454| ​

455|         def \_getFDComputedDetails(self, record : tuple):

456|                 passedTimeDelta = relativedelta(currentDate, record[4])

457|                 yearsPassed = int(passedTimeDelta.years + (passedTimeDelta.months / 12) + (passedTimeDelta.days / 365.25))

458|                 matured = False if yearsPassed < record[5] else True

459|                 value = (record[2] \* record[3] \* (record[5] if matured else yearsPassed) / 100) + record[2]

460| ​

461|                 return (yearsPassed, matured, value)

462| ​

463|         def \_printFD(self, fdName : str) -> None:

464|             if not checkFDExists(self.\_username, fdName):

465|                 print("FD with this name does not exist.")

466|                 return

467|

468|             execute(self.\_Q\_GET\_FD\_DETAILS, (self.\_username, fdName))

469|             record = crsr.fetchone()

470|             computedDetails = self.\_getFDComputedDetails(record)

471| ​

472|             print(f"Principal : {record[2]}")

473|             print(f"Interest : {record[3]}")

474|             print(f"Created : {record[4]}")

475|             print(f"Total time period (years) : {record[5]}")

476|             print(f"Time passed (years) : {computedDetails[0]}")

477|             print(f"Current value : {computedDetails[2]}")

478|             print(f"Mature date : {record[6]}")

479|             print(f"Matured? : {'Yes' if computedDetails[1] else 'No'}")

480|             print(f"Widthdrawn? : {'Yes' if record[7] else 'No'}")

481| ​

482|         def \_withdrawFD(self, fdName : str) -> None:

483|             if not checkFDExists(self.\_username, fdName):

484|                 print("FD with this name does not exist.")

485|                 return

486| ​

487|             execute(self.\_Q\_GET\_FD\_DETAILS, (self.\_username, fdName))

488|             record = crsr.fetchone()

489| ​

490|             if record[7]:

491|                 print("You have already withdrawn this FD.")

492|                 return

493|

494|             computedDetails = self.\_getFDComputedDetails(record)

495|             execute(self.\_QC\_WITHDRAW\_FD, (self.\_username, fdName))

496|             c\_changeBalance(self.\_username, computedDetails[2])

497|             c\_createUpdate(self.\_username, f"Withdrew amount {computedDetails[2]} from FD {fdName}.")

498| ​

499|             db.commit()

500| ​

501|             print(f"Withdrew amount {computedDetails[2]} from FD {fdName}.")

502|

503|         def process(self):

504|             global currentState

505| ​

506|             # display FDs

507| ​

508|             print("=============================")

509|             print("(0) Show all FDs")

510|             print("(1) View details of a particular FD")

511|             print("(2) Withdraw an FD")

512|             print("(3) Return")

513|             print()

514| ​

515|             option = intInput("(Option) -> ")

516| ​

517|             if option == 0:

518|                 execute(self.\_Q\_GET\_ALL\_FDS, (self.\_username,))

519|                 fdNames = crsr.fetchall()

520| ​

521|                 if not resultExists(fdNames):

522|                     print("You don't have any FDs yet.")

523|                     return

524|

525|                 for fdName in fdNames:

526|                     print(fdName[0])

527| ​

528|             elif option == 1:

529|                 print()

530|                 fdName = input("(Enter FD name) -> ").strip()

531|                 print()

532|                 self.\_printFD(fdName)

533| ​

534|             elif option == 2:

535|                 print()

536|                 fdName = input("(Enter FD name) -> ").strip()

537|                 print()

538|                 self.\_withdrawFD(fdName)

539| ​

540|             elif option == 3:

541|                 currentState = UnlockedState(self.\_username)

542| ​

543|             else:

544|                 print()

545|                 print("Please choose a valid option.")

546| ​

547|     class ViewUpdatesState:

548|         def \_\_init\_\_(self, username):

549|             self.\_username = username

550| ​

551|         def \_displayUpdates(self, updates):

552|             if not resultExists(updates) :

553|                 print("You have no updates for the requested query.")

554|                 return

555| ​

556|             # sort updates from most recent to last

557|             updates.sort(key = lambda x: x[2])

558|             for index, update in enumerate(updates):

559|                 baseContent, extraContent, updateDate = update

560|                 print()

561|                 print(f"({index}): {baseContent}")

562|                 print(f"Date: {updateDate}")

563|                 print(f"Comment: {extraContent}")

564| ​

565|         def process(self):

566|             global currentState

567| ​

568|             print("=============================")

569|             print("(0) View all updates")

570|             print("(1) View all updates for a day")

571|             print("(2) Return")

572|             print()

573| ​

574|             option = intInput("(Option) -> ")

575| ​

576|             if option == 0:

577|                 updates = getUpdates(self.\_username)

578|                 self.\_displayUpdates(updates)

579| ​

580|             elif option == 1:

581|                 inp = input("(Required date, in YYYY-MM-DD format) -> ")

582| ​

583|                 try:

584|                     date = dt.date.fromisoformat(inp)

585|                     updates = getUpdates(self.\_username, date)

586|                     self.\_displayUpdates(updates)

587| ​

588|                 except ValueError:

589|                     print("Invalid date.")

590| ​

591|             elif option == 2:

592|                 currentState = UnlockedState(self.\_username)

593| ​

594|             else:

595|                 print("Please choose a valid option.")

596| ​

597|     if \_\_name\_\_ == '\_\_main\_\_':

598|         currentState = LockedState()

599| ​

600|         \_Q\_GETDBCREATIONDATETIME = ("SELECT DBCreationDateTime "

601|                                     "FROM EnvInfo ;")

602| ​

603|         # Get the date and time when we created the database

604|         execute(\_Q\_GETDBCREATIONDATETIME, ())

605| ​

606|         creationDateTime = crsr.fetchone()[0]

607|         creationTime = creationDateTime.timestamp()

608|         creationDate = creationDateTime.date()

609| ​

610|         previousTime = creationTime

611|         currentDate = creationDate

612| ​

613|         while True:

614|             currentTime = time.time()

615|             elapsedDays = (currentTime - previousTime) // TIMEDELTA

616|             currentDate += dt.timedelta(days=elapsedDays)

617| ​

618|             currentState.process()

619|             previousTime = currentTime

620| ​

621| except (DataError, DatabaseError, OperationalError, NotSupportedError, IntegrityError, ProgrammingError, InternalError) as e:

622|     print("DB Error!", e)

623| ​

624| except KeyboardInterrupt:

625|     EXIT(0)

626| ​

627| except Exception as e:

628|     print("ERROR: ", e)

629|     EXIT(1)

630| ​



**SAMPLE**

**OUTPUT**

**BIBLIOGRAPHY**

* Sumita Arora Textbook Computer Science with Python Class XII