1. Examination and Evaluation of an ATM System: Identifying Software Defects and Performance Issues.

***class* Account:**

***def* \_\_init\_\_(*self*, *bal*=0):**

***self*.balance, *self*.history = *bal*, [*f*"Start: {*bal*}"]**

***def* deposit(*self*, *amt*):**

**if *amt*<=0:**

**return "Invalid"**

***self*.balance+=*amt*;**

***self*.history.append(*f*"+{*amt*} => {*self*.balance}"); return "OK"**

***def* withdraw(*self*, *amt*):**

**if *amt*<=0 or *amt*>*self*.balance:**

**return "Invalid"**

***self*.balance-=*amt*;**

***self*.history.append(*f*"-{*amt*} => {*self*.balance}");**

**return "OK"**

***def* statement(*self*):**

**[print(h) for h in *self*.history]**

**acc=Account(100);**

**print(acc.deposit(50));**

**print(acc.withdraw(30));**

**acc.statement()**

1. **Case Study on ARMS Portals: Analyzing Requirements and Developing Positive Test Scenarios**

**class ARMSportal:**

**def \_\_init\_\_(self):**

**self.users = {"192211654" : "hussain"}**

**def login(self, u , p):**

**return "Login Success" if self.users.get(u)==p else "Login Falied"**

**portal = ARMSportal()**

**print("Test case: Valid Login")**

**u = input("Enter username:")**

**p = input("Enter password:")**

**result = portal.login(u, p)**

**print("Expected : Login Success")**

**print("Actual:", result)**

**print("Result:", "PASS" if result=="Login Success" else "FAIL")**

1. **Case Study on ARMS Portals: Analyzing Requirements and Developing Negative Test Scenarios**

**class ARMSPORTAL:**

**def \_\_init\_\_(self): self.users = {"19211654":"mohammad"}**

**def login(self, u, p): return "Login Success" if self.users.get(u)==p else "invalid username or password"**

**portal = ARMSPORTAL()**

**print("Test Case: Invalid Login")**

**u = input("Enter username:")**

**p = input("Enter password:")**

**result = portal.login(u,p)**

**print("Expected: Invalid username or password")**

**print("Actual:", result)**

**print("Result", "PASS" if result=="invalid username or password" else "FAIL")**

1. Case Study on e-Commerce Applications: Analyzing Requirements and Creating Positive Test Scenarios

class ECOMMERCEAPP:

def \_\_init\_\_(self): self.products=["Laptop","phone"]; self.cart=[]; self.order=[]

def add\_to\_cart(self, item): self.cart.append(item); return f"{item} added"

def checkout(self):

if not self.cart: return "cart empty"

self.order.extend(self.cart); self.cart.clear(); return "Order placed"

def payment(self,amount): return "payment successful" if amount>0 else "payment failed"

app = ECOMMERCEAPP()

print(app.add\_to\_cart("Laptop"))

print(app.payment(20000))

result = app.checkout()

print("Expected: Order placed")

print("Actual:", result)

print("Result:", "PASS" if result=="Order placed" else "FAIL" )

5.Case Study on e-Commerce Applications: Analyzing Requirements and Creating Negative Test Scenarios

class ECommerceApp:

def \_\_init\_\_(self): self.products=["Laptop","Phone"]; self.cart=[]; self.orders=[]

def add\_to\_cart(self, item): self.cart.append(item); return f"{item} added"

def checkout(self): return "Please add items to your cart" if not self.cart else "Order placed"

def payment(self, amount): return "Payment failed" if amount<=0 else "Payment successful"

app = ECommerceApp()

print("Test: Checkout with empty cart")

result = app.checkout()

print("Expected: Please add items to your cart")

print("Actual:", result)

print("Result:", "PASS" if result=="Please add items to your cart" else "FAIL")

print("\nTest: Invalid payment")

result2 = app.payment(-500)

print("Expected: Payment failed")

print("Actual:", result2)

print("Result:", "PASS" if result2=="Payment failed" else "FAIL")

1. .Case Study on Food Delivery Applications: Analyzing Requirements and Creating Positive Test Scenarios

cclass FoodDeliveryApp:

def \_\_init\_\_(self): self.menu=["Pizza","Burger"]; self.cart=[]; self.orders=[]

def add\_to\_cart(self, item): self.cart.append(item); return f"{item} added to cart"

def payment(self, amount): return "Payment successful" if amount>0 else "Payment failed"

def checkout(self):

if not self.cart: return "Cart empty"

self.orders.extend(self.cart); self.cart.clear(); return "Order confirmed. Delivery in 30 mins"

app = FoodDeliveryApp()

print(app.add\_to\_cart("Pizza"))

print(app.payment(250))

result = app.checkout()

print("Expected: Order confirmed. Delivery in 30 mins")

print("Actual:", result)

print("Result:", "PASS" if result=="Order confirmed. Delivery in 30 mins" else "FAIL")

1. **Case Study on Food Delivery Applications: Analyzing Requirements and Creating  Negative Test Scenarios**

*class* FoodDeliveryApp:

*def* \_\_init\_\_(*self*): *self*.menu=["Pizza","Burger"]; *self*.cart=[]; *self*.orders=[]

*def* add\_to\_cart(*self*, *item*): *self*.cart.append(*item*); return *f*"{*item*} added"

*def* payment(*self*, *amount*): return "Invalid payment method" if *amount*<=0 else "Payment successful"

*def* checkout(*self*): return "Your cart is empty. Please add items" if not *self*.cart else "Order confirmed"

app = FoodDeliveryApp()

print("Test: Checkout without items")

result = app.checkout()

print("Expected: Your cart is empty. Please add items")

print("Actual:", result)

print("Result:", "PASS" if result=="Your cart is empty. Please add items" else "FAIL")

print("\nTest: Invalid payment")

result2 = app.payment(-100)

print("Expected: Invalid payment method")

print("Actual:", result2)

print("Result:", "PASS" if result2=="Invalid payment method" else "FAIL")

8. Case Study on Banking Applications: Analyzing Requirements and Developing Positive Test Scenarios

class BankingApp:

def \_\_init\_\_(self): self.accounts={"user1":1000,"user2":500}; self.transactions=[]

def login(self, u, p): return "Login Success" if u in self.accounts else "Login Failed"

def transfer(self, from\_acc, to\_acc, amt):

if self.accounts[from\_acc]>=amt: self.accounts[from\_acc]-=amt; self.accounts[to\_acc]+=amt

self.transactions.append(f"{from\_acc} -> {to\_acc}: {amt}"); return "Transfer Successful"

def balance(self, acc): return f"Balance: {self.accounts[acc]}"

app = BankingApp()

print(app.login("user1","pass"))

print(app.transfer("user1","user2",300))

print(app.balance("user1"))

print(app.balance("user2"))

print("Transactions:", app.transactions)

9.Case Study on Banking Applications: Analyzing Requirements and Developing Negative Test Scenarios

class BankingApp:

def \_\_init\_\_(self): self.accounts={"user1":1000,"user2":500}; self.transactions=[]

def login(self, u, p): return "Login Success" if u in self.accounts and p=="pass" else "Invalid username or password"

def transfer(self, from\_acc, to\_acc, amt):

if from\_acc not in self.accounts or to\_acc not in self.accounts: return "Invalid account"

if self.accounts[from\_acc]<amt: return "Insufficient funds. Please check your balance"

self.accounts[from\_acc]-=amt; self.accounts[to\_acc]+=amt; self.transactions.append(f"{from\_acc}->{to\_acc}:{amt}"); return "Transfer Successful"

app = BankingApp()

print("Test: Invalid login")

print(app.login("user1","wrongpass"))

print("Test: Transfer exceeding balance")

print(app.transfer("user1","user2",2000))

10. Test Plan Development for Library Management System

class LMS:

def \_\_init\_\_(self): self.users={"admin":"admin123"}

def login(self, u, p): return "Login successful, dashboard opened" if self.users.get(u)==p else "Invalid username or password"

lms = LMS()

print("Test: Valid login")

print(lms.login("admin","admin123"))

print("Test: Invalid login")

print(lms.login("admin","wrongpassword"))

print("Test: Non-existent user")

print(lms.login("user1","pass"))