**Scenario:**  
You are testing an online shopping platform where users can apply discount coupons based on their total purchase amount. The discount policy is as follows:

* **No discount** for amounts **below $50**
* **10% discount** for amounts **between $50 and $100** (inclusive)
* **20% discount** for amounts **above $100**

Your task is to apply **Equivalence Partitioning (EP)** to identify valid test cases.

**Step 1: Identify Equivalence Classes**

Based on the given conditions, we can divide the input domain into three **equivalence partitions**:

| **Partition** | **Input Range** | **Expected Output** |
| --- | --- | --- |
| Class 1 (No discount) | Amount < 50 | 0% Discount |
| Class 2 (10% discount) | 50 ≤ Amount ≤ 100 | 10% Discount |
| Class 3 (20% discount) | Amount > 100 | 20% Discount |

**Step 2: Choose Representative Test Cases**

From each equivalence class, we select **one representative test case**:

1. **Class 1 (No Discount)**
   * Test case: **Amount = 30**
   * Expected result: **0% discount**
2. **Class 2 (10% Discount)**
   * Test case: **Amount = 75**
   * Expected result: **10% discount**
3. **Class 3 (20% Discount)**
   * Test case: **Amount = 120**
   * Expected result: **20% discount**

**Step 3: Verify with Boundary Values (Optional)**

For better coverage, we can also test boundary values:

* **Class 1 (No discount)**
  + Test case: **Amount = 49** → Expected: **0% discount**
* **Class 2 (10% discount)**
  + Test case: **Amount = 50** → Expected: **10% discount**
  + Test case: **Amount = 100** → Expected: **10% discount**
* **Class 3 (20% discount)**
  + Test case: **Amount = 101** → Expected: **20% discount**

**Final Test Cases for Execution**

| **Test Case** | **Input Amount** | **Expected Output** |
| --- | --- | --- |
| 1 | 30 | 0% Discount |
| 2 | 75 | 10% Discount |
| 3 | 120 | 20% Discount |
| 4 | 49 | 0% Discount |
| 5 | 50 | 10% Discount |
| 6 | 100 | 10% Discount |
| 7 | 101 | 20% Discount |

#### **Scenario 2:**

A bank's **ATM withdrawal system** allows users to withdraw cash under the following conditions:

1. **Minimum withdrawal amount is $100**
2. **Maximum withdrawal amount is $10,000 per transaction**
3. The withdrawal amount must be **a multiple of $100**
4. The account balance must be **sufficient** to complete the transaction

### ****Step 1: Identify Equivalence Classes****

| **Condition** | **Valid Partition** | **Invalid Partitions** |
| --- | --- | --- |
| **Minimum Amount** | $100 or more | Less than $100 |
| **Maximum Amount** | $10,000 or less | More than $10,000 |
| **Multiples of $100** | 100, 200, 300, ... | Any non-multiple of 100 (e.g., 150, 275) |
| **Sufficient Balance** | Balance ≥ Withdrawal | Balance < Withdrawal |

### ****Step 2: Select Representative Test Cases****

| **Test Case** | **Withdrawal Amount** | **Balance** | **Valid/Invalid?** | **Expected Result** |
| --- | --- | --- | --- | --- |
| 1 | **50** | $500 | ❌ Invalid (Min limit) | Rejected |
| 2 | **100** | $500 | ✅ Valid | Approved |
| 3 | **10,000** | $15,000 | ✅ Valid | Approved |
| 4 | **10,500** | $20,000 | ❌ Invalid (Max limit) | Rejected |
| 5 | **275** | $500 | ❌ Invalid (Not a multiple of 100) | Rejected |
| 6 | **1,000** | $900 | ❌ Invalid (Insufficient balance) | Rejected |
| 7 | **5,000** | $6,000 | ✅ Valid | Approved |

### ****Step 3: Additional Boundary Testing (Optional)****

For a more thorough test, we also check the **boundary values**:

| **Test Case** | **Withdrawal Amount** | **Balance** | **Expected Result** |
| --- | --- | --- | --- |
| 8 | **99** | $500 | ❌ Rejected (Just below min limit) |
| 9 | **100** | $500 | ✅ Approved (Exactly min limit) |
| 10 | **9,900** | $20,000 | ✅ Approved (Just below max limit) |
| 11 | **10,000** | $20,000 | ✅ Approved (Exactly max limit) |
| 12 | **10,001** | $20,000 | ❌ Rejected (Just above max limit) |

### ****Cause-Effect Analysis Example: ATM Withdrawal System****

#### **Scenario:**

A bank's ATM withdrawal system allows customers to withdraw money based on these conditions:

1. The **minimum withdrawal amount** is **$100** (**C1**)
2. The **maximum withdrawal amount** is **$10,000 per transaction** (**C2**)
3. The withdrawal amount must be **a multiple of $100** (**C3**)
4. The account must have **sufficient balance** for the requested amount (**C4**)
5. The ATM must have **sufficient cash** available to dispense the requested amount (**C5**)

If all conditions are met, the withdrawal is **successful** (**E1**). Otherwise, the transaction is **rejected** (**E2**).

### ****Step 1: Identify Causes & Effects****

| **Causes (Inputs)** | **Effects (Outputs)** |
| --- | --- |
| **C1:** Withdrawal ≥ $100 | **E1:** Successful withdrawal |
| **C2:** Withdrawal ≤ $10,000 | **E2:** Rejected (if not met) |
| **C3:** Withdrawal is a multiple of $100 | **E2:** Rejected (if not met) |
| **C4:** Sufficient account balance | **E2:** Rejected (if not met) |
| **C5:** Sufficient ATM cash available | **E2:** Rejected (if not met) |

### ****Step 2: Cause-Effect Table****

| **Test Case** | **C1: Min Amount ($100+)** | **C2: Max Amount ($10,000 max)** | **C3: Multiple of $100** | **C4: Sufficient Balance** | **C5: ATM Has Cash** | **Expected Outcome** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | ✅ | ✅ | ✅ | ✅ | ✅ | **Approved** |
| 2 | ❌ (50) | ✅ | ✅ | ✅ | ✅ | **Rejected (Below min amount)** |
| 3 | ✅ | ❌ (12,000) | ✅ | ✅ | ✅ | **Rejected (Above max amount)** |
| 4 | ✅ | ✅ | ❌ (275) | ✅ | ✅ | **Rejected (Not a multiple of $100)** |
| 5 | ✅ | ✅ | ✅ | ❌ (Low balance) | ✅ | **Rejected (Insufficient balance)** |
| 6 | ✅ | ✅ | ✅ | ✅ | ❌ (ATM low cash) | **Rejected (ATM can’t dispense)** |

### ****Step 3: Interpret the Results****

* **Any failure in one or more conditions (C1–C5) leads to rejection** (**E2**).
* **Only if all conditions are met, the transaction is successful** (**E1**).

**Q1. What is the main goal of integration testing?**  
a) To test the individual units of a system  
b) To test interactions between different modules  
c) To test end-user functionality  
d) To validate system performance

**Answer:** b)

Q2. **Which of the following is a common approach to integration testing?**  
a) Top-Down  
b) Bottom-Up  
c) Big Bang  
d) All of the above

**Answer:** d)

Q3. **What is a "stub" in integration testing?**  
a) A temporary module that simulates lower-level components  
b) A real module that replaces a defective one  
c) A test case used for performance testing  
d) A tool used for automating integration tests

**Answer:** a)

**Q4. Which integration testing approach starts testing from the lowest-level modules?**  
a) Top-Down Integration  
b) Bottom-Up Integration  
c) Big Bang Integration  
d) Regression Testing

**Answer:** b)

Q5. **Which type of integration testing is most suitable when all components are developed simultaneously?**  
a) Incremental Integration Testing  
b) Big Bang Integration Testing  
c) Top-Down Integration Testing  
d) Bottom-Up Integration Testing

**Answer:** b)

Q6. **Which of the following is a key benefit of unit testing?**  
a) Ensures complete end-to-end system validation  
b) Increases development speed by eliminating all bugs  
c) Helps detect defects early at the module level  
d) Primarily focuses on verifying the user interface

**Answer:** c)

**Q7. What is the primary reason for using stubs and drivers in integration testing?**  
a) To replace missing components during early testing  
b) To test real-time performance  
c) To automate regression testing  
d) To simulate user interactions

**Answer:** a)

Q8. **In which scenario is "Big Bang Integration Testing" least effective?**  
a) When all modules are completed at the same time  
b) When debugging needs to be done quickly  
c) When high dependencies exist between modules  
d) When incremental development is followed

**Answer:** d)

Q9. **Which of the following statements is true about incremental integration testing?**  
a) It requires all modules to be completed before testing begins  
b) It is performed after system testing  
c) Modules are integrated and tested one by one  
d) It does not use stubs or drivers**:**

**Answer** c)

Q10. **Which of the following is a disadvantage of Top-Down Integration Testing?**  
a) Critical modules are tested early  
b) Stubs may not accurately simulate lower-level module behavior  
c) It requires the entire system to be ready before testing  
d) It does not require test case design

**Answer:** b

Q11. **Scenario:** A developer is working on an e-commerce application and writes unit tests for the payment processing function. The function takes in an order total and applies a discount if the user has a valid promo code.  
**Which of the following test cases should be included in unit testing?**  
a) Applying a valid promo code and verifying the discounted total  
b) Checking whether the bank processes the payment successfully  
c) Verifying if the user receives an order confirmation email  
d) Testing how multiple users interact with the system at the same time

**Answer:** a)

Q12. **Scenario:** A banking application has a function that calculates monthly interest for savings accounts based on the balance. The function should apply a 3% interest rate for balances above $10,000 and a 2% interest rate for lower balances.  
**Which unit test is most appropriate?**  
a) Test the function with a balance of $5,000 and check if 2% interest is applied  
b) Verify that the function updates all account balances in the database  
c) Ensure that the interest calculation function works when integrated with the reporting module  
d) Check if the user receives a monthly interest notification email

**Answer:** a)

Q13. **Scenario:** A developer is testing a login function that checks for correct username and password combinations.  
**Which of the following test cases should be included in unit testing?**  
a) Valid username and correct password  
b) Invalid username and incorrect password  
c) SQL injection attack attempt  
d) All of the above

**Answer:** d)

Q14. **Scenario:** A company is developing an online ticket booking system with the following modules:

* **M1:** User Login
* **M2:** Ticket Selection
* **M3:** Payment Processing
* **M4:** Ticket Confirmation

The company decides to use **Bottom-Up Integration Testing**.  
**Which of the following should be tested first?**  
a) User Login (M1)  
b) Payment Processing (M3)  
c) Ticket Confirmation (M4)  
d) Ticket Selection (M2)

**Answer:** b)

Q15. **Scenario:** A developer is testing an integration between a shopping cart module and a payment gateway. The payment gateway processes transactions asynchronously and returns a success or failure response.  
**Which is the most critical integration test case?**  
a) Ensuring that an item can be added to the cart  
b) Checking if the payment gateway correctly handles a failed transaction  
c) Verifying the shopping cart layout on different browsers  
d) Ensuring the customer receives a promotional email after a purchase

**Answer:** b)

Q16. **Scenario:** A hospital management system consists of three integrated modules:

* **M1:** Patient Registration
* **M2:** Doctor Appointment Scheduling
* **M3:** Prescription & Billing

The hospital wants to follow a **Top-Down Integration Testing** approach.  
**Which of the following should be tested using a stub?**  
a) Patient Registration (M1)  
b) Doctor Appointment Scheduling (M2)  
c) Prescription & Billing (M3)  
d) None, as all modules are tested simultaneously

**Answer:** c)

Q17. **Case Study:** A software company is developing an online food delivery platform. The system has three core modules:

* **Order Placement (M1)**
* **Restaurant Processing (M2)**
* **Delivery Tracking (M3)**

**Question:** The development team decides to use **Incremental Integration Testing** instead of Big Bang Testing.  
**Which of the following benefits does Incremental Testing provide?**  
a) Detects defects earlier in the integration process  
b) Allows all modules to be tested simultaneously  
c) Reduces the number of required test cases  
d) Avoids the need for stubs or drivers

**Answer:** a)

Q18. **Case Study:** A financial services company has built a stock trading platform where users can:

* View real-time stock prices (**M1**)
* Place buy/sell orders (**M2**)
* Receive trade confirmations (**M3**)

The company follows **Top-Down Integration Testing** but finds that stock prices are not updating correctly in the user dashboard.  
**What is the most likely reason?**  
a) A stub was used instead of the real stock price API  
b) The trade confirmation module is not working correctly  
c) The buy/sell order system is not processing correctly  
d) The application has not been tested on all mobile devices

**Answer:** a)

#### Q19. **Scenario: E-commerce Website Checkout**

A customer tries to check out an order, but when they enter an invalid credit card number, the system does not display an error message and proceeds to the payment page.

**Which black box testing technique should be used to verify proper validation of the credit card number?**  
a) Equivalence Partitioning  
b) White Box Testing  
c) Statement Coverage Testing  
d) Mutation Testing

**Answer:** a)

Q20. An ATM allows customers to withdraw between $100 and $5,000 in multiples of $100. A user tries to withdraw $90, and the system processes the request incorrectly instead of rejecting it.

**Which black box testing technique can be applied to verify withdrawal limits?**  
a) State Transition Testing  
b) Boundary Value Analysis  
c) Control Flow Testing  
d) Code Coverage Testing

**Answer:** b)

Q21. A banking application has a password reset feature. Users can enter a new password, but it must be at least 8 characters long and contain a mix of uppercase, lowercase, numbers, and special characters. Some users report that weak passwords are still being accepted.

**Which black box testing approach is most suitable to test this functionality?**  
a) Cause-Effect Graphing  
b) State Transition Testing  
c) Boundary Value Analysis  
d) Fuzz Testing

**Answer:** a)

Q22. A mobile banking app has the following transaction rules:

* Maximum transfer per transaction: $10,000
* Minimum transfer per transaction: $1
* Daily transfer limit: $50,000

A user successfully transfers $10,100 in a single transaction, violating the system rules.

**Which black box testing method should be applied to find such issues?**  
a) Equivalence Partitioning  
b) Boundary Value Analysis  
c) Static Testing  
d) Path Testing

**Answer:** b)

**Q23. In white box testing, statement coverage ensures that:**

A) Every possible path in the code is executed at least once  
B) Every decision in the code has been evaluated both true and false  
C) Every executable statement in the code is executed at least once  
D) Every condition in a decision statement is tested

Answer: C

**Q24. Which of the following is true about branch coverage?**

A) It ensures that all individual conditions within a decision are tested  
B) It ensures that all possible branches (true and false) of each decision point are executed  
C) It tests all independent paths in the program at least once  
D) It ensures that all statements are executed at least once

**Answer:** B)

**Q25. A developer is testing the following function:**

def check\_even(n):

if n % 2 == 0:

print("Even")

print("Done")

How many test cases are required to achieve **100% statement coverage**?

A) 1  
B) 2  
C) 3  
D) 4

**Answer:** A

**Q26. Consider the following code:**

def check\_number(n):

if n > 0:

print("Positive")

else:

print("Negative or Zero")

Which test cases are required to achieve **100% branch coverage**?

A) n = 1  
B) n = -1  
C) n = 0 and n = 1  
D) n = -1 and n = 1

**Answer:** D

**Q27. Given the following code:**

def is\_eligible(age, citizen):

if age >= 18 and citizen == "Yes":

return "Eligible"

else:

return "Not Eligible"

Which test cases achieve **100% condition coverage**?

A) (18, "Yes"), (17, "Yes")  
B) (18, "Yes"), (18, "No")  
C) (18, "Yes"), (17, "No")  
D) (18, "Yes"), (17, "Yes"), (18, "No"), (17, "No")

**Answer:** D

**Q28. Consider the following function:**

def check\_value(x):

if x > 10:

print("Greater")

elif x < 5:

print("Smaller")

else:

print("Between")

How many test cases are required to achieve **100% path coverage**?

A) 2  
B) 3  
C) 4  
D) 5

**Answer:** B

**Q29. A function processes a list of numbers using a loop:**

def process\_list(lst):

total = 0

for num in lst:

total += num

return total

Which test cases ensure **loop coverage**?

A) lst = [], lst = [5], lst = [1, 2, 3]  
B) lst = [0], lst = [5, 10]  
C) lst = [1, 2, 3], lst = [4, 5]  
D) lst = [], lst = [1]

**Answer:** A

**Q30. Consider the following Python function:**

python

CopyEdit

def check\_number(n):

if n > 0:

print("Positive")

elif n < 0:

print("Negative")

else:

print("Zero")

What is the **Cyclomatic Complexity** of this function?

A) 1  
B) 2  
C) 3  
D) 4

**Answer:** C

**Q31. Which of the following increases the Cyclomatic Complexity of a program?**

A) More print statements  
B) More function calls  
C) More conditional statements (if, switch, while)  
D) More comments in the code

**Answer:** C

**Q32. Consider the following function:**

def process\_items(items):

for item in items:

if item % 2 == 0:

print("Even")

else:

print("Odd")

What is the **Cyclomatic Complexity** of this function?

A) 1  
B) 2  
C) 3  
D) 4

**Answer:** C

**Q33. What is a major disadvantage of a high Cyclomatic Complexity in a codebase?**

A) It reduces code execution speed  
B) It increases the difficulty of debugging and maintaining the code  
C) It reduces memory usage  
D) It improves code readability

**Answer:** B

**Q34. How does a switch-case statement affect Cyclomatic Complexity?**

A) It increases CC by the number of case statements  
B) It doesn’t impact CC  
C) It reduces CC  
D) It increases CC by 1 regardless of the number of case statements

**Answer:** A

**Q35. Which of the following is an effective way to reduce Cyclomatic Complexity in a large function?**

A) Replacing if-else statements with loops  
B) Using **function decomposition** (splitting into smaller functions)  
C) Adding more comments to explain complex logic  
D) Increasing the number of variables

**Answer:** B

**Q36. Consider the following function:**

python

CopyEdit

def analyze\_number(n):

if n > 0:

if n % 2 == 0:

print("Positive Even")

else:

print("Positive Odd")

elif n < 0:

print("Negative")

else:

print("Zero")

What is the **Cyclomatic Complexity** of this function?

A) 2  
B) 3  
C) 4  
D) 5

**Answer:** D

**Q37.** You analyze a function with a **Cyclomatic Complexity of 6**. What does this imply about the number of **independent paths** in the function?

A) 4  
B) 5  
C) 6  
D) 7

**Answer:** C)

Q38. How does a **nested loop** affect **Cyclomatic Complexity**?

A) It does not affect CC  
B) It increases CC by 1 for each nested loop  
C) It increases CC by 2 for each nested loop  
D) It increases CC exponentially

**Answer:** B

Q39. A function contains:

* **10 edges**
* **7 nodes**
* **1 connected component**

What is its **Cyclomatic Complexity**?

A) 2  
B) 3  
C) 4  
D) 5

**Answer:** C

**Q40.** Consider the following function with a switch statement:

def menu(option):

switch(option):

case 1: print("Option 1")

case 2: print("Option 2")

case 3: print("Option 3")

default: print("Invalid Option")

What is the **Cyclomatic Complexity** of this function?

A) 2  
B) 3  
C) 4  
D) 5

**Answer:** D

**Q41.** A function contains **4 if statements**, each independent (not nested). What is the **maximum possible Cyclomatic Complexity**?

A) 4  
B) 5  
C) 6  
D) 8

**Answer:** B

**Q42.** If a developer **copies and pastes** the same logic multiple times in a function, how does it affect **Cyclomatic Complexity**?

A) It **increases** CC  
B) It **decreases** CC  
C) It **does not change** CC  
D) It makes CC unpredictable

**Answer:** C)