Q1. process age test cases

In [ ]:

```
def process_age(age):
    if age < 0:
        return "Invalid"
    if age < 18:
        return "Minor"
    if age > 65:
        return "Senior"
    if age < 0: # Unreachable!
        return "Error"
    return "Adult"

def test_process_age():
    assert process_age(-5) == "Invalid"
    assert process_age(10) == "Minor"
    assert process_age(70) == "Senior"
    assert process_age(32) == "Adult"
    print("All tests passed")

test_process_age()
```

Q1. sum until negative test cases

In [ ]:

```
# Assume input array has array size 3
def sum_until_negative(numbers):
    total = 0
    i = 0
    while i < len(numbers) and numbers[i] >= 0:
        total += numbers[i]
        i += 1
    return total


def test_sum_until_negative():
    assert sum_until_negative([]) == 0
    assert sum_until_negative([1,2,3]) == 6
    assert sum_until_negative([1,-3,4]) == 1
    assert sum_until_negative([-1]) == 0
    print("All tests passed")

test_sum_until_negative()
```

Q1. classify sequnece test cases

In [ ]:

```
def classify_sequence(numbers):
    if len(numbers) == 0:
        return "Empty"
    count = 0
    i = 0
    while i < len(numbers) and i < 5: # Process at most 5 numbers
        if numbers[i] > 0:
            count += 1
        i += 1

    if count == 0:
        return "AllNonPositive"
    elif count == i:
        return "AllPositive"
    else:
        return "Mixed"


def test_classify_sequence():
    assert classify_sequence([])
    assert classify_sequence([1, 2, 3])
    assert classify_sequence([-1, -5, -4])
    assert classify_sequence([1, 2, -1])
    print("All tests passed")

test_classify_sequence()
```

Q1. all positive smt

In [ ]:

```
from z3 import *

solver = Solver()
x1, x2, x3 = Ints('x1 x2 x3')
solver.add(x1 > 0, x2 > 0, x3 > 0)

if solver.check() == sat:
    model = solver.model()
    solution = [model[x1], model[x2], model[x3]]
    print("Solution found:", solution)
```

```
else:
    print("No solution exists.")
```

Q2. Cloud Alert System

- Simulate Incoming Metrics
- Trigger Alerts Based on Thresholds
- Notification Handling
- Alert Resolution
- Logging & Reporting

In [ ]:

```
import time
import random
import logging
from datetime import datetime, timedelta
import numpy as np

logging.basicConfig(level=logging.INFO, format='[%(asctime)s] %(message)s')

# Alert severity and thresholds
ALERT_THRESHOLDS = {
    "P0": {"latency": 2000, "failure_rate": 10, "interval": 2},
    "P1": {"latency": 1000, "failure_rate": 5, "interval": 12},
    "P2": {"latency": 500, "failure_rate": 2, "interval": 48},
}

# Emails
target_email = "team@example.com"
skip_level_email = "boss@example.com"

active_alerts = {}
log_records = []
log_retention_days = 90

def generate_metrics():
    latency = np.random.poisson(750)
    failure_rate = np.random.poisson(3) / 100
    return latency, failure_rate * 100

def determine_alert(latency, failure_rate):
    for severity, params in ALERT_THRESHOLDS.items():
        if latency > params["latency"] or failure_rate > params["failure_rate"]:
            return severity
    return None

def send_email(recipient, subject, message):
    logging.info(f"{subject} - {message} <EMAIL to {recipient}>")

def log_system_status(latency, failure_rate):
    log_records.append((datetime.now(), latency, failure_rate))
    logging.info(f"INFO: Latency: {latency}ms, Failure Rate: {failure_rate:.2f}%")

def resolve_alerts():
    if active_alerts:
        for alert_id in list(active_alerts.keys()):
            logging.info(f"INFO: Commit {random.randint(1000, 9999)} submitted")
            del active_alerts[alert_id]
            logging.info(f"INFO: Alert {alert_id} resolved.")

def check_alerts():
    now = datetime.now()
    for alert_id, alert_data in list(active_alerts.items()):
        severity, timestamp, notified, escalation_time = alert_data
        resend_interval = ALERT_THRESHOLDS[severity]["interval"]

        if now >= timestamp + timedelta(hours=resend_interval) and not notified:
            logging.info(f"ALERT: Resending {severity} alert (Still unresolved)")
            active_alerts[alert_id] = (severity, now, False, escalation_time)

        if now >= escalation_time:
            logging.info(f"ESCALATION: {severity} alert unresolved for too long. Send a notification to skip-level boss.")
            del active_alerts[alert_id]

def clean_old_logs():
    cutoff = datetime.now() - timedelta(days=log_retention_days)
    global log_records
    log_records = [record for record in log_records if record[0] > cutoff]

def run_monitoring_system(duration_minutes=60):
    start_time = datetime.now()
    while (datetime.now() - start_time).total_seconds() < duration_minutes * 60:
        latency, failure_rate = generate_metrics()
        log_system_status(latency, failure_rate)

        alert_severity = determine_alert(latency, failure_rate)
        if alert_severity:
            alert_id = f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
            if alert_id not in active_alerts:
                logging.info(f"{alert_id} {alert_severity} Alert Triggered.")
                send_email(target_email, f"ALERT: {alert_severity} Triggered", "Immediate action required.")
                escalation_time = datetime.now() + timedelta(hours=5 * ALERT_THRESHOLDS[alert_severity]["interval"])
                active_alerts[alert_id] = (alert_severity, datetime.now(), False, escalation_time)

        # resolving an issue simulation
        if random.random() < 0.3:
            resolve_alerts()

        check_alerts()
        clean_old_logs()

        time.sleep(10)

if __name__ == "__main__":
    run_monitoring_system(duration_minutes=60)
```