NAME: TEST

Q1. Describe the Software Development Life Cycle (SDLC) and compare the Waterfall model with Agile methodology in detail.

Answer: The Software Development Life Cycle (SDLC) is a systematic process used by industries to design, develop, and test high-quality software. The primary goal is to produce software that meets or exceeds customer expectations within reaching time and cost estimates. It consists of several distinct stages: planning, requirements analysis, design, implementation, testing, deployment, and maintenance. Each phase has its own set of deliverables and entry/exit criteria. The Waterfall model is a linear and sequential approach where each phase must be completed before the next one begins. It is highly disciplined and easy to manage, but it lacks flexibility, making it difficult to accommodate changes once the development is underway. In contrast, Agile methodology follows an iterative and incremental approach. It focuses on continuous improvement, flexibility, and customer feedback. Agile breaks the project into small cycles called sprints, allowing teams to deliver functional software components frequently. While Waterfall is suitable for projects with well-defined requirements and fixed scopes, Agile is preferred for complex projects where requirements are expected to evolve over time. Agile promotes cross-functional team collaboration and rapid response to market changes, which is essential in modern software engineering environments. The choice between these models depends on project scale, budget constraints, and the level of stakeholder involvement required throughout the development process.

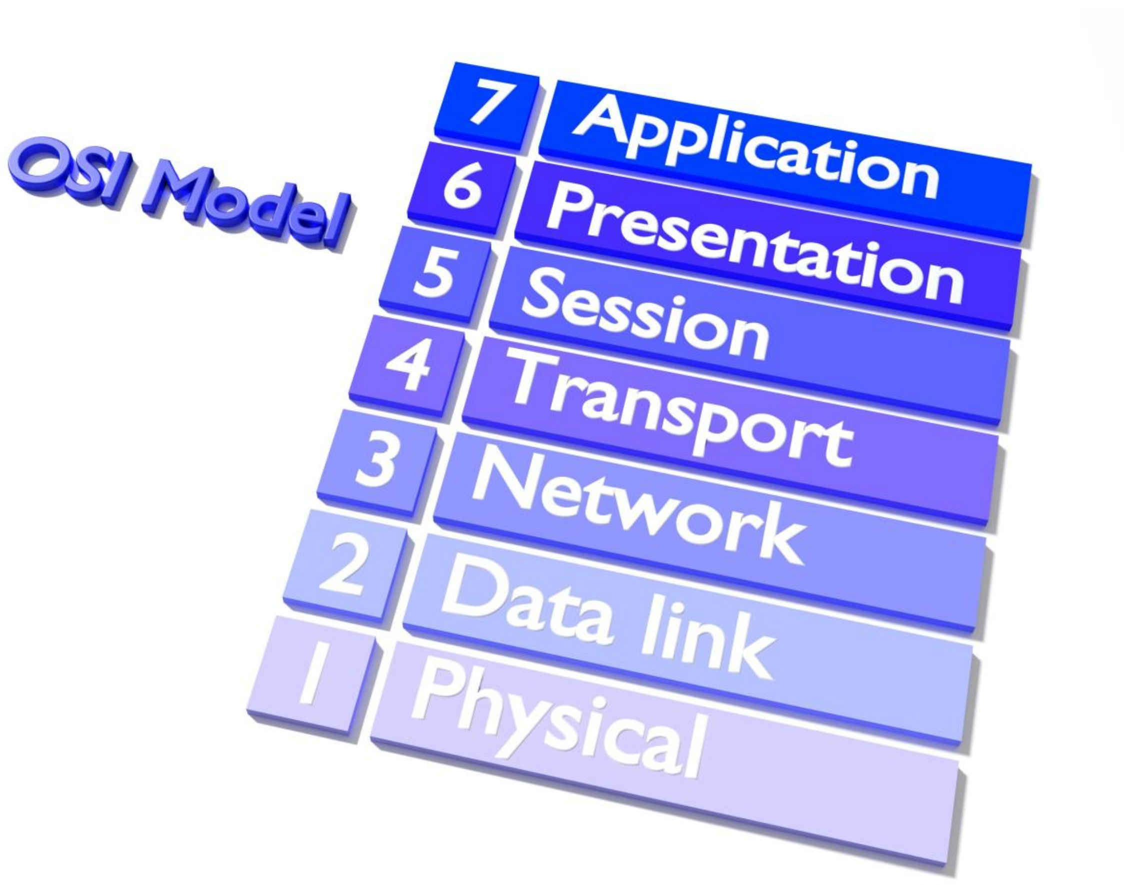
Q2. Explain the fundamental concepts of Relational Database Management Systems (RDBMS) and the importance of ACID properties.

Answer: A Relational Database Management System (RDBMS) is a type of database management system that stores data in tables with rows and columns. Tables are linked to each other using keys, such as Primary Keys and Foreign Keys, to ensure data integrity and minimize redundancy. The core strength of an RDBMS lies in its ability to handle structured data and complex queries through Structured Query Language (SQL). To ensure the reliability of transactions within a database, the ACID properties must be strictly followed. ACID stands for Atomicity, Consistency, Isolation, and Durability. Atomicity ensures that a transaction is treated as a single unit, which either succeeds completely or fails entirely; there is no partial completion. Consistency ensures that a transaction brings the database from one valid state to another, maintaining all predefined rules and constraints. Isolation ensures that concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially. This prevents transactions from interfering with each other. Finally, Durability guarantees that once a transaction has been committed, it will remain so, even in the case of a power loss or system crash. These properties are critical for financial systems and applications where data accuracy is paramount. Without ACID compliance, databases would be prone to corruption and inconsistent states during high-traffic operations.

Q3. Discuss the architecture of Operating Systems, focusing on the differences between Monolithic and Microkernel designs.

Answer: The architecture of an Operating System (OS) defines how the kernel interacts with the hardware and how system services are delivered to user applications. The kernel is the core component that manages system resources like CPU, memory, and I/O devices. In a Monolithic kernel architecture, all OS services, including file management, memory management, and device drivers, run in the same address space as the kernel. This design offers high performance because communication between services is direct and fast. However, it suffers from poor maintainability and reliability; if one service fails, the entire system might crash. On the other hand, the Microkernel architecture minimizes the kernel's responsibilities by moving most non-essential services, such as device drivers and file systems, into user space as separate processes. Communication between these services is handled through Inter-Process Communication (IPC). This approach significantly enhances system modularity and security. If a service in user space crashes, it does not affect the core kernel, making the system more robust. However, the overhead of IPC can lead to slower performance compared to Monolithic systems. Modern operating systems often use a hybrid approach to balance performance and modularity. Choosing the right architecture depends on the specific requirements of the hardware, such as whether it is a general-purpose computer or a specialized embedded system.

Q4. Elaborate on the OSI Model and explain the function of each of its seven layers in computer networking.



Shutterstock

Explore

Answer: The Open Systems Interconnection (OSI) model is a conceptual framework used to understand and standardize the functions of a telecommunication or computing system without regard to its internal structure and technology. It divides network communication into seven distinct layers, each serving a specific role. Starting from the bottom, the Physical Layer deals with the physical connection and transmission of raw bits over a medium. The Data Link Layer handles node-to-node data transfer and error correction. Above that, the Network Layer is responsible for packet forwarding and routing through intermediate routers. The Transport Layer ensures end-to-end communication, flow control, and reliability using protocols like TCP and UDP. The Session Layer manages the dialogues between computers, establishing and terminating connections. The Presentation Layer acts as a data translator, handling encryption, decryption, and data compression. Finally, the Application Layer provides the interface for end-user applications like web browsers and email clients to access network services. By separating functions into layers, the OSI model allows vendors to develop interoperable hardware and software. It also simplifies troubleshooting, as network administrators can isolate problems to a specific layer. While the TCP/IP model is more commonly used in practice today, the OSI model remains the gold standard for educational and theoretical understanding of how different network components interact.

Q5. Analyze the benefits and challenges of Cloud Computing service models: IaaS, PaaS, and SaaS.

Answer: Cloud Computing has revolutionized the way businesses manage IT infrastructure by providing on-demand access to computing resources over the internet. The three primary service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS provides the most flexibility by offering virtualized computing resources like servers, storage, and networking. Users are responsible for managing the operating system, middleware, and applications, which gives them full control but requires significant technical expertise. PaaS simplifies the development process by providing a platform that includes the operating system, development tools, and database management systems. This allows developers to focus solely on coding and deploying applications without worrying about the underlying infrastructure. SaaS delivers fully functional software applications directly to users through a web browser. It eliminates the need for installation and maintenance on local devices but offers the least amount of customization. While cloud computing offers benefits like scalability, cost-efficiency, and global accessibility, it also presents challenges. Security and data privacy remain top concerns, as sensitive information is stored on third-party servers. Additionally, organizations may face vendor lock-in, making it difficult to migrate data or services to another provider. Understanding the trade-offs between these models is essential for organizations to optimize their cloud strategy.

SECTION 2: CODING / LOGIC QUESTIONS

Q6. Implement a Python class for a Simple JWT (JSON Web Token) simulation for user authentication.

Answer: import base64 import json import hmac import hashlib import time

class SimpleJWT: def **init**(self, secret_key): self.secret_key = secret_key self.algorithm = 'HS256'

```
def create_token(self, payload):

    header = {"alg": self.algorithm, "typ": "JWT"}

    header_encoded = self._base64_url_encode(json.dumps(header))


    # Add expiration to payload

    payload['exp'] = int(time.time()) + 3600

    payload_encoded = self._base64_url_encode(json.dumps(payload))
```

```
        signature_base = f"{header_encoded}.{payload_encoded}"
        signature = self._generate_signature(signature_base)

        return f"{signature_base}.{signature}"

    def _base64_url_encode(self, data):
        return base64.urlsafe_b64encode(data.encode()).decode().rstrip("=")

    def _generate_signature(self, data):
        signature = hmac.new(
            self.secret_key.encode(),
            data.encode(),
            hashlib.sha256
        ).digest()
        return base64.urlsafe_b64encode(signature).decode().rstrip("=")

    def verify_token(self, token):
        try:
            parts = token.split('.')
            if len(parts) != 3:
                return False, "Invalid token format"

            header_encoded, payload_encoded, signature_provided = parts
            signature_base = f"{header_encoded}.{payload_encoded}"
            expected_signature = self._generate_signature(signature_base)

            if not hmac.compare_digest(signature_provided, expected_signature):
                return False, "Signature mismatch"

            payload = json.loads(base64.urlsafe_b64decode(payload_encoded + "=="))
            if time.time() > payload.get('exp', 0):
```

```
        return False, "Token expired"

      return True, payload
    except Exception as e:
      return False, str(e)
```

Example Usage

```
auth_system = SimpleJWT("super_secret_key_123") user_payload = {"user_id": 101, "role": "admin"}
new_token = auth_system.create_token(user_payload) print("Generated Token:", new_token)
is_valid, data = auth_system.verify_token(new_token) print("Verification Result:", is_valid, data)
```

Q7. Write a Python script to perform CRUD operations on a local JSON file acting as a database.

Answer: import json import os

class JSONDatabase: def **init**(self, filename): self.filename = filename if not os.path.exists(self.filename): with open(self.filename, 'w') as f: json.dump([], f)

```
def _read_data(self):
  with open(self.filename, 'r') as f:
    return json.load(f)

def _write_data(self, data):
  with open(self.filename, 'w') as f:
    json.dump(data, f, indent=4)

def create_record(self, item):
  data = self._read_data()
  item['id'] = len(data) + 1
  data.append(item)
  self._write_data(data)
  return item

def read_records(self):
  return self._read_data()

def update_record(self, item_id, new_values):
  data = self._read_data()
  updated = False
  for item in data:
    if item['id'] == item_id:
        item.update(new_values)
        updated = True
        break
  if updated:
    self._write_data(data)
  return updated


def delete_record(self, item_id):
  data = self._read_data()
  initial_length = len(data)
  data = [item for item in data if item['id'] != item_id]
  if len(data) < initial_length:
    self._write_data(data)
    return True
  return False
```

Testing the logic

db = JSONDatabase("test_db.json") db.create_record({"name": "Alice", "email": "alice@example.com"}) db.create_record({"name": "Bob", "email": "bob@example.com"}) print("Records after creation:", db.read_records()) db.update_record(1, {"email": "alice_updated@example.com"}) db.delete_record(2) print("Final records:", db.read_records())

Q8. Create a Python implementation of a Linked List with methods for insertion, deletion, and searching.

Answer: class Node: def **init**(self, data): self.data = data self.next = None

class LinkedList: def **init**(self): self.head = None

```
def insert_at_end(self, data):
    new_node = Node(data)
    if not self.head:
        self.head = new_node
        return
    last = self.head
    while last.next:
        last = last.next
    last.next = new_node

def delete_node(self, key):
    temp = self.head
    if temp is not None:
        if temp.data == key:
            self.head = temp.next
            temp = None
            return
    while temp is not None:
        if temp.data == key:
            break
        prev = temp
        temp = temp.next
    if temp is None:
        return
    prev.next = temp.next
    temp = None

def search(self, key):
    current = self.head
    while current:
        if current.data == key:
            return True
        current = current.next
    return False
```

```
def display(self):
    elements = []
    current = self.head
    while current:
        elements.append(str(current.data))
        current = current.next
    print(" -> ".join(elements))
```

Driver code

llist = LinkedList() llist.insert_at_end(10) llist.insert_at_end(20) llist.insert_at_end(30) llist.display() print("Search 20:", llist.search(20)) llist.delete_node(20) llist.display() print("Search 20 after deletion:", llist.search(20))

Q9. Develop a Python function to simulate a basic Bank Account system with exception handling for overdrafts.

Answer: class InsufficientFundsError(Exception): pass

class BankAccount: def **init**(self, owner, balance=0.0): self.owner = owner self.balance = balance self.transaction_history = []

```
def deposit(self, amount):
    if amount <= 0:
        print("Invalid deposit amount")
        return
    self.balance += amount
    self.transaction_history.append(f"Deposited: ${amount}")
    print(f"Successfully deposited ${amount}")


def withdraw(self, amount):
    try:
        if amount > self.balance:
            raise InsufficientFundsError(f"Overdraft Attempted: Current balance is ${self.balance}")
        if amount <= 0:
            print("Invalid withdrawal amount")
            return
        self.balance -= amount
        self.transaction_history.append(f"Withdrew: ${amount}")

        print(f"Successfully withdrew ${amount}")

    except InsufficientFundsError as e:

        print(f"Error: {e}")


def get_statement(self):

    print(f"\nAccount Statement for {self.owner}")

    for transaction in self.transaction_history:

        print(transaction)

    print(f"Final Balance: ${self.balance}\n")
```

Execution sequence

my_account = BankAccount("John Doe", 500.0) my_account.deposit(200) my_account.withdraw(100) my_account.withdraw(1000) # Should trigger exception my_account.get_statement()

Q10. Write a recursive Python program to solve the Fibonacci sequence up to 'n' terms with a memoization decorator.

Answer: def memoize_fib(f): cache = {} def helper(x): if x not in cache: cache[x] = f(x) return cache[x] return helper

@memoize_fib def fibonacci(n): if n == 0: return 0 elif n == 1: return 1 else: return fibonacci(n-1) + fibonacci(n-2)

def generate_fib_sequence(limit): sequence = [] for i in range(limit): sequence.append(fibonacci(i)) return sequence

Testing efficiency

start_time = time.time() n_terms = 35 result = generate_fib_sequence(n_terms) end_time = time.time()

print(f"Fibonacci sequence up to {n_terms} terms:") print(result) print(f"Calculated in {end_time - start_time:.6f} seconds")

SECTION 3: PROGRAMMING LANGUAGE QUESTIONS

Q11. What is the difference between shallow copy and deep copy in Python?

Answer: A shallow copy creates a new object, but it inserts references to the objects found in the original. If the original object contains nested objects (like a list inside a list), the shallow copy will still point to those same nested objects. Changes made to nested objects in the copy will reflect in the original. A deep copy, however, creates a new object and recursively adds copies of the nested objects found in the original. This means the copy is entirely independent of the original. In Python, the copy module provides the copy() and deepcopy() functions for these purposes.

Q12. Explain the concept of 'Self' in Python classes.

Answer: In Python, self is a conventional name used as the first parameter of instance methods in a class. It represents the instance of the object itself. When a method is called on an object, Python automatically passes the object reference as the first argument. This allows the method to access and modify the object's attributes and other methods. Without self, the class would not know which specific object's data it should be operating on. Note that self is not a keyword; any name can be used, but following the convention is crucial for readability.

Q13. How do decorators work in Python?

Answer: Decorators are a powerful tool in Python that allow programmers to modify or extend the behavior of a function or class without permanently changing its source code. A decorator is itself a function that takes another function as an argument and returns a new function with added functionality. They are commonly used for logging, access control, and timing. The @decorator_name syntax is used above a function definition as a shorthand to apply the decorator.

Q14. Describe the purpose of *args and **kwargs in function definitions.

Answer: *args and **kwargs allow a function to accept a variable number of arguments. *args is used to pass a non-keyworded, variable-length argument list (as a tuple), allowing the function to handle more arguments than specified. **kwargs allows you to pass keyworded, variable-length arguments (as a dictionary). This provides great flexibility when the exact number of inputs is unknown at the time of writing the function, common in wrappers and API development.

Q15. What are Python Generators and how do they differ from regular functions?

Answer: Generators are special types of functions that return an iterable set of items one at a time using the yield keyword instead of return. Unlike regular functions that compute a value and terminate, a generator pauses its execution and saves its state, allowing it to resume later. This makes generators highly memory-efficient, especially when dealing with large datasets, as they generate values on-the-fly (lazy evaluation) rather than storing the entire list in memory.