MODULE 1 – OVERVIEW OF THE IT INDUSTRY

---

1. What is a Program?

- **Theoretical Overview:** A program constitutes an organized set of instructions written in a programming language that a computer system can interpret and execute. Its operation involves accepting inputs, executing algorithmic processes, and generating outputs in alignment with defined objectives.
- **Laboratory Exercise:**

```python
# Python
print("Hello, World!")
```

```c
// C
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Comparison: Python provides syntactic simplicity and abstraction from memory management. In contrast, C requires explicit structuring, affording greater control over system resources.

---

2. What is Programming?

- **Theoretical Overview:** Programming is the intellectual discipline of conceptualizing, implementing, testing, and refining algorithmic solutions encoded in a formal language, facilitating computational problem-solving.

---

3. Types of Programming Languages

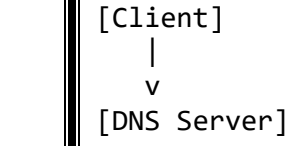
- **Theoretical Overview:** High-level languages, such as Python and Java, offer abstraction from hardware intricacies, emphasizing readability and developer productivity. Conversely, low-level languages, including Assembly and Machine Code, provide granular control at the hardware level, enabling performance optimization and direct resource manipulation.

---

4. The World Wide Web and Internet Mechanics

- **Laboratory Exercise:**
- **Theoretical Overview:** Web communication hinges on client-server interactions mediated by protocols like HTTP. Clients initiate requests, while servers fulfill them by providing relevant content or services.

Diagram:



```
   |
   v
[Web Server]
   |
   v
[HTTP Response]
   |
   v
[Browser Rendering]
```

---

5. Network Layers in Client-Server Architectures

- **Laboratory Exercise:**

```
from http.server import BaseHTTPRequestHandler, HTTPServer
class SimpleHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello from Server")
HTTPServer(("", 8000), SimpleHandler).serve_forever()
```

- **Theoretical Overview:**

The TCP/IP model delineates four integral layers—Application, Transport, Internet, and Network Access—each responsible for distinct aspects of end-to-end data communication.

---

6. Client and Server Communication

- **Theoretical Overview:** In distributed computing, clients issue requests which are processed by servers, thereby facilitating dynamic data exchange over network protocols.

---

7. Internet Connection Modalities

- **Laboratory Exercise:** | Connection Type | Pros | Cons | |————-|————————|————————-| | Broadband | Stable, moderately fast | Shared bandwidth | | Fiber Optic | High speed and reliability | Cost-intensive infrastructure | | Satellite | Remote accessibility | Latency, weather dependent |
- **Theoretical Overview:** While broadband employs electrical transmission via copper wires, fiber-optic networks utilize photonic signals, resulting in superior speed and reduced signal degradation.

---

8. Network Protocols

- **Laboratory Exercise:**

```
curl http://example.com
curl ftp://speedtest.tele2.net
```

- **Theoretical Overview:** HTTP facilitates web content transfer in plaintext, whereas HTTPS employs SSL/TLS encryption to secure transmitted data.

---

9. Application Security

- **Laboratory Exercise:**

1. SQL Injection → Employ prepared statements
2. Cross-Site Scripting (XSS) → Implement input validation

3. Cross-Site Request Forgery (CSRF) → Utilize anti-CSRF tokens

- **Theoretical Overview:** Encryption encodes data to maintain confidentiality and integrity during transmission and storage, forming a critical pillar of application security.

---

10. Software Applications and Classification

- **Laboratory Exercise:** | Software | Type | |———–|———–-| | Chrome | Application | | Windows | System | | MS Word | Application | | Antivirus | Utility | | VLC Player | Application |
- **Theoretical Overview:** System software orchestrates fundamental operations and hardware management, while application software facilitates user-directed functionalities.

---

11. Software Architecture

- **Laboratory Exercise:**
- **Theoretical Overview:** Modularity in architecture ensures compartmentalization, thereby enhancing scalability, maintainability, and unit testing.

---

12. Layered Architecture Case Study

- **Laboratory Exercise:** Presentation: HTML/CSS interface
  Logic: Server-side processing (e.g., PHP, Python)
  Data Access: SQL-based database interaction
- **Theoretical Overview:** Layered structures isolate concerns, allowing independent development and optimization of individual software components.

---

13. Software Environments

- **Laboratory Exercise:** Deploy an Ubuntu-based virtual machine equipped with development tools (e.g., VS Code, Python)
- **Theoretical Overview:** Development environments simulate the application runtime and provide IDEs, compilers, and debugging utilities essential for pre-deployment testing.

---

14. Source Code

- **Laboratory Exercise:** Create and push a code repository to GitHub
- **Theoretical Overview:** Source code is the human-readable logic behind applications, which compilers translate into machine code—binary instructions executed by the CPU.

---

15. GitHub Basics

- **Laboratory Exercise:**

```
git init
git add .
git commit -m "Initial commit"
git push origin main
```

- **Theoretical Overview:** Version control systems such as Git manage codebase evolution, facilitate collaboration, and preserve developmental history.

---

16. Student Collaboration via GitHub

- **Laboratory Exercise:** Execute a team-based project using GitHub’s collaborative features.

- **Theoretical Overview:** GitHub introduces students to industry-standard practices for project management, source control, and peer collaboration.

---

17. Software Taxonomy

- **Laboratory Exercise:** | Software | Category | |————-|————————| | Windows | System | | Chrome | Application | | MS Word | Application | | WinRAR | Utility | | Antivirus | Utility |
- **Theoretical Overview:** Open-source software promotes transparency and community-driven development. Proprietary software restricts access to source code and redistributive rights.

---

18. Git Fundamentals

- **Laboratory Exercise:** Practice branching, merging, and cloning operations
- **Theoretical Overview:** Git facilitates distributed version control, enabling concurrent feature development and streamlined integration workflows.

---

19. Application Software in Business

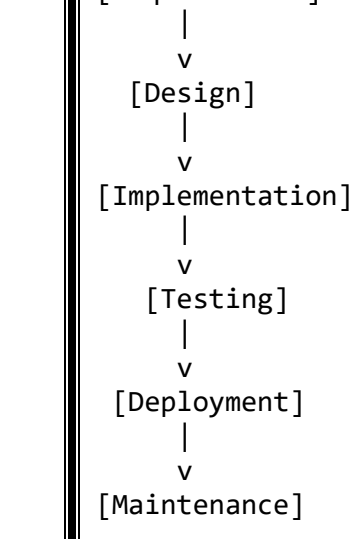
- **Laboratory Exercise:** | Software | Purpose | |————-|————————| | MS Word | Text Processing | | Excel | Data Analysis | | Chrome | Internet Browsing | | Zoom | Video Conferencing | | Photoshop | Multimedia Editing |
- **Theoretical Overview:** Application software augments organizational efficiency through automation, data processing, and communication facilitation.

---

20. Software Development Life Cycle (SDLC)

- **Laboratory Exercise:**
- **Theoretical Overview:** The SDLC encompasses phases from eliciting user needs to system decommissioning, ensuring structured and traceable software evolution.

Flowchart:



---

21. Requirements Engineering

- **Laboratory Exercise:** Specification Document for Library Management:
  Functionalities: Book Addition, Search, Issue, Return
  User Roles: Admin, Student
- **Theoretical Overview:** The requirements phase formalizes system objectives, minimizing ambiguity and preventing scope creep.

---

22. Software Analysis

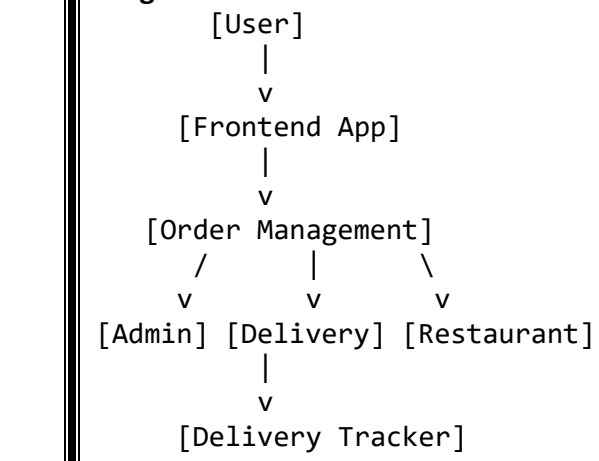
- **Laboratory Exercise:** Analyze functional modules of an e-commerce platform: Product Search, Cart Management, Payment Gateway
- **Theoretical Overview:** Software analysis dissects user and system requirements, laying the foundation for architectural design.

---

23. System Design

- **Laboratory Exercise:**

Diagram:



- **Theoretical Overview:** System design encapsulates architectural schematics, data models, and interface specifications that dictate overall solution structure.

---

24. Software Testing

- **Laboratory Exercise:** | Test Case | Input | Expected Output | |———–|———–|————————| | TC1 | Add(2,3) | 5 | | TC2 | Divide(4,0)| Exception/Error |
- **Theoretical Overview:** Testing uncovers software anomalies, verifies compliance with requirements, and ensures robustness.

---

25. Software Maintenance

- **Laboratory Exercise:** Document the security patch update for a mobile banking application
- **Theoretical Overview:** Maintenance types:
- Corrective: Defect Resolution
- Adaptive: Environmental Modifications
- Perfective: Performance Enhancements

---

26. Development Paradigms

- **Theoretical Overview:** Web applications operate cross-platform via browsers, while desktop applications are confined to specific operating systems and local installations.

---

27. Web Applications

- **Theoretical Overview:** Benefits include ubiquitous accessibility, centralized updates, and platform independence.

---

28. Interface Design

- **Theoretical Overview:** UI/UX design governs the interactive and aesthetic dimensions of software, critically influencing user satisfaction and engagement.

---

29. Mobile Applications

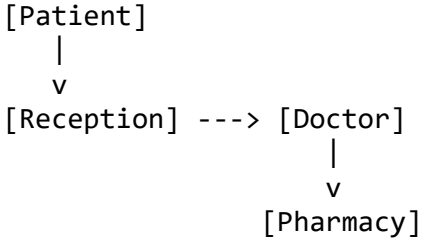
- **Theoretical Overview:** Native applications are tailored to individual platforms for optimal performance; hybrid apps employ cross-platform frameworks but may sacrifice performance.

---

30. Data Flow Diagrams (DFD)

- **Laboratory Exercise:**

DFD:



- **Theoretical Overview:** DFDs illustrate data movement and transformation, essential for system modeling and stakeholder communication.

---

31. Desktop Applications

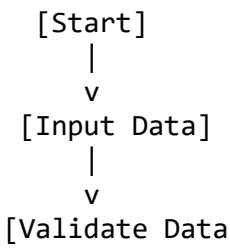
- **Laboratory Exercise:** Develop a GUI-based calculator using Python's Tkinter library
- **Theoretical Overview:** Advantages include offline functionality and direct system access; disadvantages entail limited portability and update complexity.

---

32. Flowcharts in System Design

- **Laboratory Exercise:**

Flowchart:



```
    |
    v
[Submit Form]
    |
    v
  [End]
```

- **Theoretical Overview:** Flowcharts render logical constructs graphically, facilitating algorithm comprehension and debugging.