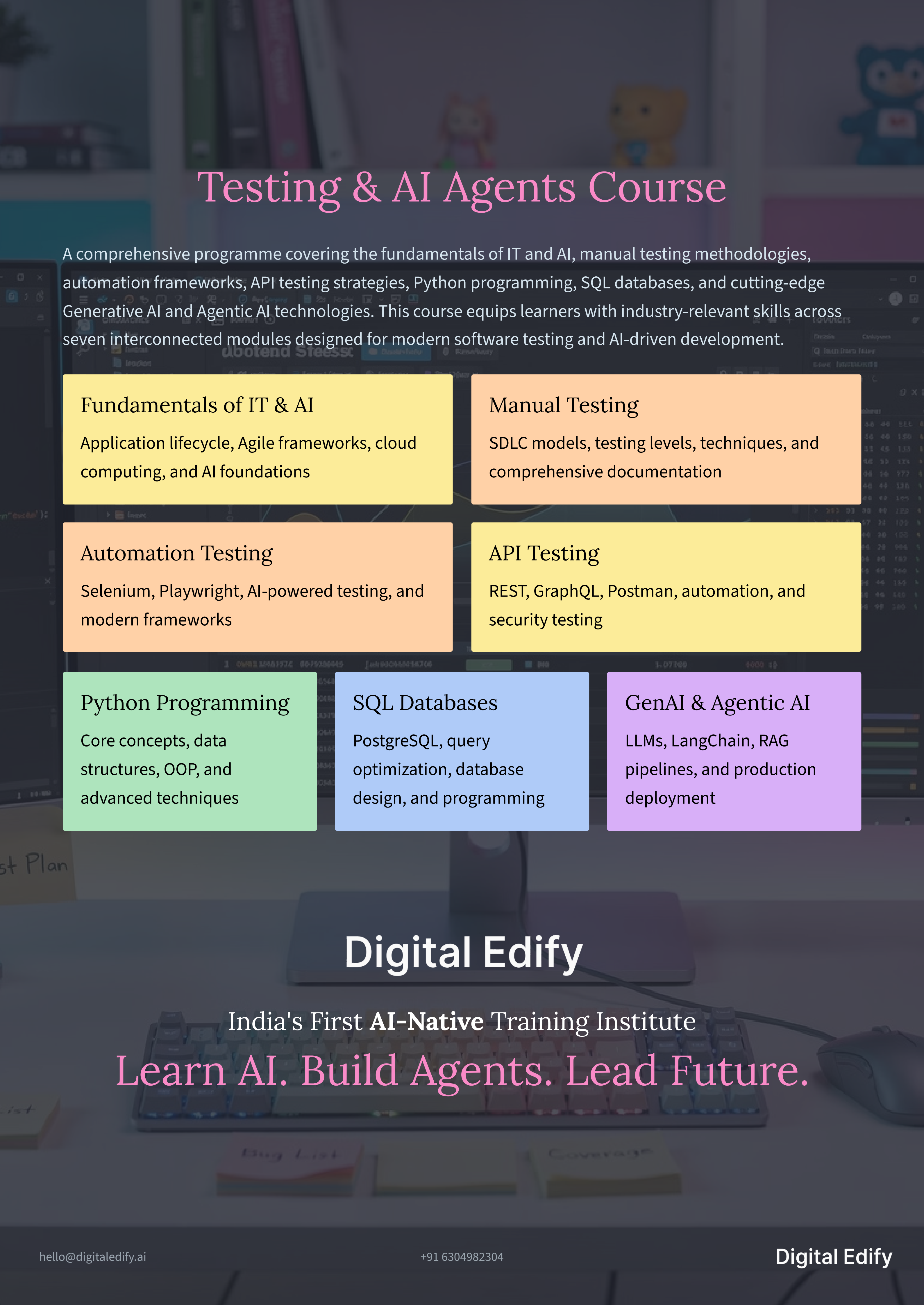


Testing & AI Agents Course

A comprehensive programme covering the fundamentals of IT and AI, manual testing methodologies, automation frameworks, API testing strategies, Python programming, SQL databases, and cutting-edge Generative AI and Agentic AI technologies. This course equips learners with industry-relevant skills across seven interconnected modules designed for modern software testing and AI-driven development.

Fundamentals of IT & AI

Application lifecycle, Agile frameworks, cloud computing, and AI foundations

Manual Testing

SDLC models, testing levels, techniques, and comprehensive documentation

Automation Testing

Selenium, Playwright, AI-powered testing, and modern frameworks

API Testing

REST, GraphQL, Postman, automation, and security testing

Python Programming

Core concepts, data structures, OOP, and advanced techniques

SQL Databases

PostgreSQL, query optimization, database design, and programming

GenAI & Agentic AI

LLMs, LangChain, RAG pipelines, and production deployment

Digital Edify

India's First **AI-Native** Training Institute

Learn AI. Build Agents. Lead Future.

hello@digitaledify.ai

+91 6304982304

Digital Edify

About Digital Edify

India's #1 Training Institute for the AI Era

Established: 2016	**Headquarters:** Hyderabad, Telangana	**Reach:** Global (Online + Offline)

The Transformation Narrative

Digital Edify has evolved from a premium training institute in the Automation Era to an AI-first organisation leading the Agentic AI revolution. Since 2016, we've transformed over 100,000 professionals and built partnerships with more than 1,000 industry leaders. Our journey reflects the technological evolution of our time—from traditional job placement to career transformation, and now to building AI-native professionals who will shape the future of work.

1

Automation Era (2016-2023)

Premium Training Institute focused on job placement with 100K+ students trained

2

AI Revolution (2024-2025)

AI-Powered Training with industry-AI integration and career transformation focus

3

Agentic AI Leadership (2026+)

AI First Institute building AI-Native Professionals with 1 Million AI-Native Vision

> "We started in the Automation Era. We evolved through the AI Revolution. Now, we're leading the Agentic AI Future—with 100,000+ professionals already transformed and 1,000+ industry partners trusting our graduates."

Vision & Mission

Vision

"To Create 1 Million AI-Native Professionals Who Will Build the Agentic Future of Work"

Mission

"We transform learners into AI-native professionals through industry-aligned programmes that integrate Agentic AI into every discipline—from development to data science to enterprise platforms."

Course Highlights

Section 1: Fundamentals of IT & AI

Build a strong foundation in application development, Agile practices, cloud computing, and core AI concepts with real-world business use cases.

Section 2: Manual Testing

Master end-to-end manual testing concepts including SDLC, test levels, functional and non-functional testing, test design techniques, Agile testing, and industry tools.

Section 3: Automation Testing

Learn modern test automation using Selenium, Playwright, AI-powered testing, CI/CD pipelines, and real-world automation frameworks.

Section 4: API Testing

Gain hands-on expertise in API testing with Postman and Python automation, covering REST, GraphQL, security, performance, and CI/CD integration.

Section 5: Python for AI & Testing

Develop strong Python programming skills for automation, data handling, testing frameworks, and AI-driven testing workflows.

Section 6: SQL for AI & Testing

Understand database fundamentals, advanced SQL querying, optimization, and database validation techniques for testing and analytics.

Section 7: Generative AI & Agentic AI

Design, build, and deploy intelligent AI systems using LLMs, prompt engineering, RAG pipelines, agentic workflows, and production-grade AI architectures.

Application Life Cycle Management

Understanding applications forms the foundation of effective testing. This module explores application types, web technologies, and the complete Software Development Life Cycle (SDLC), providing essential context for testing professionals to comprehend how software is built and maintained.

01

Planning

Define project scope, objectives, and resource requirements

02

Analysis

Gather and document detailed requirements from stakeholders

03

Design

Create system architecture and technical specifications

04

Implementation

Develop code following design specifications and standards

05

Testing

Verify functionality, performance, and quality standards

06

Deployment

Release software to production environments

07

Maintenance

Provide ongoing support, updates, and enhancements

Frontend Technologies

- HTML for structure
- CSS for styling
- JavaScript for interactivity
- React for modern interfaces

Backend Technologies

- Python for versatile development
- Java for enterprise solutions
- Node.js for scalable services

SQL Databases

MySQL and PostgreSQL for structured, relational data storage

NoSQL Databases

MongoDB for flexible, document-based data management

Module 2: Agile & Scrum Framework

Modern software development demands flexibility and iterative approaches. This module contrasts traditional Waterfall methodology with Agile principles, diving deep into the Scrum framework that has become the industry standard for managing complex projects through collaboration, transparency, and continuous improvement.

Product Owner

Maximises product value and manages the product backlog

Scrum Master

Facilitates Scrum processes and removes impediments

Development Team

Cross-functional professionals delivering increments

Sprint Planning

Team selects work from product backlog for the upcoming sprint

Daily Scrum

15-minute synchronisation meeting to plan the day's work

Sprint Review

Demonstrate completed work to stakeholders for feedback

Product Backlog

Ordered list of everything needed in the product

Sprint Backlog

Selected items plus plan for delivering the increment

Increment

Sum of all completed backlog items meeting Definition of Done

User Stories capture requirements from an end-user perspective, typically following the format: "As a [user type], I want [goal] so that [benefit]." Acceptance criteria define when a story is complete, whilst epics group related stories into larger initiatives.

Module 3: Computing & Data Foundations

Computing power and cloud infrastructure underpin modern applications. Understanding CPU and GPU capabilities, alongside cloud service models, enables testers to comprehend performance requirements and deployment architectures that influence testing strategies and environments.

CPU Architecture

Central Processing Units handle general-purpose computing tasks, executing sequential operations with high single-thread performance for traditional application logic.

GPU Computing

Graphics Processing Units excel at parallel processing, accelerating AI workloads, machine learning training, and data-intensive computations through thousands of simultaneous operations.

Infrastructure as a Service (IaaS)

Virtualised computing resources including servers, storage, and networking infrastructure managed by cloud providers

Platform as a Service (PaaS)

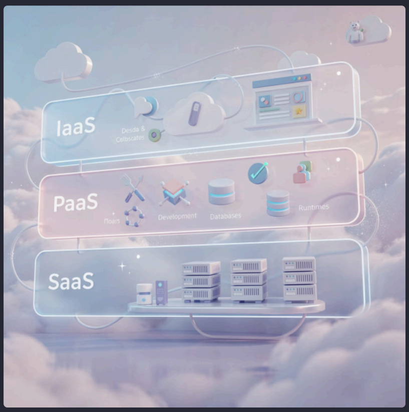
Development and deployment platforms with built-in tools, databases, and middleware for application creation

Software as a Service (SaaS)

Complete applications delivered over the internet, accessible through browsers without local installation

Cloud Computing Benefits

- Scalability on demand
- Reduced infrastructure costs
- Global accessibility
- Automatic updates and maintenance
- Enhanced collaboration capabilities



Module 4: Introduction to AI Technologies

Artificial Intelligence

Systems that simulate human intelligence and decision-making

Machine Learning

Algorithms that learn patterns from data without explicit programming

Deep Learning

Neural networks with multiple layers processing complex patterns

Generative AI

Models creating new content from learned patterns and prompts

Large Language Models

LLMs like GPT, Claude, and Gemini process and generate human-like text, understanding context, answering questions, and assisting with complex reasoning tasks across multiple domains.

- Natural language understanding
- Text generation and completion
- Translation and summarisation
- Code generation and debugging

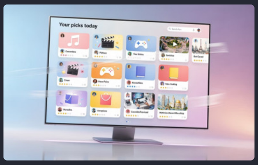
Image Generation Models

AI systems create original images from text descriptions, enabling creative applications in design, marketing, and content creation through sophisticated neural architectures.

- Text-to-image synthesis
- Style transfer and editing
- Image enhancement and restoration
- Creative design assistance

Personal Assistants

Voice-activated AI helps with scheduling, reminders, and information retrieval



Recommendations

Personalised content and product suggestions based on user behaviour

Autonomous Systems

Self-driving vehicles and robotics powered by computer vision and ML

Module 5: Real-World Applications

Enterprise applications solve specific business challenges across industries. Understanding CRM, HRMS, retail, e-commerce, and healthcare systems provides testers with domain knowledge essential for effective test planning, scenario creation, and quality assurance in specialised environments.

Customer Relationship Management

CRM systems manage customer interactions, sales pipelines, marketing campaigns, and service requests, centralising customer data to improve relationships and drive revenue growth.

Human Resource Management

HRMS platforms handle recruitment, employee records, payroll, performance reviews, and benefits administration, streamlining HR processes and workforce management.

Retail & E-Commerce

Digital commerce platforms enable online shopping, inventory management, payment processing, and order fulfilment. Key features include:

- Product catalogues and search
- Shopping cart and checkout
- Payment gateway integration
- Order tracking and delivery
- Customer reviews and ratings
- Personalised recommendations

Healthcare Applications

Medical systems manage patient records, appointments, prescriptions, and clinical workflows whilst ensuring regulatory compliance. Critical components:

- Electronic Health Records (EHR)
- Appointment scheduling
- Prescription management
- Laboratory information systems
- Telemedicine platforms
- HIPAA compliance features

Software Testing Fundamentals

1

System Software

Operating systems and utilities managing hardware resources

2

Programming Software

Development tools, compilers, and IDEs for creating applications

3

Application Software

End-user programmes solving specific business or personal needs

Quality Assurance vs Quality Control

Errors, Bugs, Defects & Failures

1

Requirement Analysis

Review requirements and identify testable features

2

Test Planning

Define test strategy, scope, and resource allocation

3

Test Case Development

Create detailed test cases and test data

4

Environment Setup

Configure test environments and tools

5

Test Execution

Run tests and log defects

6

Test Closure

Evaluate completion criteria and document lessons

Module 2: SDLC Models & Testing Approaches

Different development methodologies require tailored testing strategies. This module examines Waterfall and V-Model approaches, exploring verification and validation phases, static testing techniques, and the distinctions between white box, black box, and grey box testing methodologies.

Waterfall Model

Sequential phases with testing after development completion. Best for stable requirements but inflexible to changes.

V-Model

Verification and validation at each development stage. Testing activities parallel development phases.

Verification Activities

- **BRS**: Business Requirements Specification
- **SRS**: Software Requirements Specification
- **HLD**: High-Level Design documents
- **LLD**: Low-Level Design documents

Static Testing Techniques

- **Review**: Informal examination of documents
- **Walkthrough**: Author-led presentation to team
- **Inspection**: Formal, structured defect detection

White Box Testing

Internal structure testing with code visibility. Testers examine logic, paths, and conditions. Requires programming knowledge. Used for unit and integration testing.

Black Box Testing

Functional testing without code knowledge. Focus on inputs, outputs, and behaviour. Tests against specifications. Most common approach for system testing.

Grey Box Testing

Hybrid approach with partial code knowledge. Combines functional and structural testing. Useful for integration and security testing scenarios.

Module 3: Testing Levels: Unit & Integration

1 Unit Testing Fundamentals

Developers test individual functions, methods, or classes in isolation. Automated unit tests provide rapid feedback, catch regressions early, and document expected behaviour through executable specifications.

2 Unit Testing Techniques

- **Basis Path Testing**: Cover all independent paths through code
- **Conditional Coverage**: Test all boolean conditions
- **Loop Coverage**: Verify loop boundaries and iterations

JUnit (Java)

Industry-standard framework for Java unit testing with annotations and assertions

NUnit (.NET)

Comprehensive testing framework for .NET applications and C# code

pytest (Python)

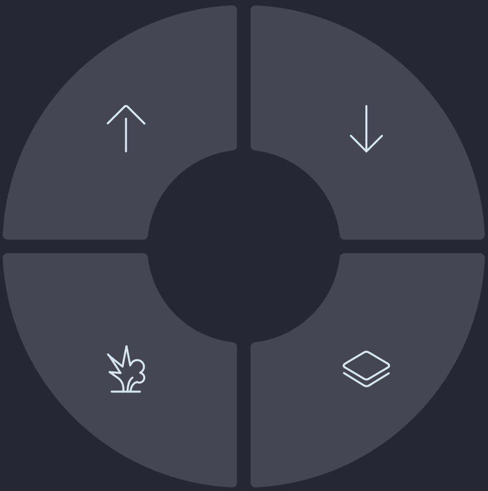
Powerful, flexible Python testing framework with fixture support

Top-Down Integration

Test from top-level modules downward, using stubs for lower modules

Bottom-Up Integration

Test from lowest modules upward, using drivers for higher modules



Big Bang Integration

Test all modules together at once, faster but harder to debug

Sandwich Integration

Hybrid approach testing top and bottom simultaneously, meeting in middle

Stubs are dummy modules simulating lower-level components not yet developed. **Drivers** are temporary modules calling components under test when higher-level modules aren't ready. Both enable testing before complete system integration.

Module 4: Testing Levels: System & Acceptance

Functional Testing

Verify features work according to specifications

Non-Functional Testing

Assess performance, security, and usability

GUI Testing

Comprehensive verification of user interface elements:

- Visual design consistency and branding
- Button, link, and form functionality
- Navigation flow and menu structure
- Content accuracy and spelling
- Responsive design across devices
- Accessibility compliance (WCAG)

Usability Testing

Evaluate user experience and satisfaction:

- Intuitive navigation and workflows
- Clear labelling and instructions
- Efficient task completion
- Error message clarity
- Learning curve assessment
- User satisfaction metrics

Alpha Testing

Internal testing by QA team or employees in a controlled environment before external release. Identifies major issues and validates core functionality.

Beta Testing

External testing by real users in production-like environments. Gathers feedback on usability, performance, and real-world scenarios before general availability.

Production Release

Software deployed to all users after successful acceptance testing and stakeholder approval. Monitoring continues post-release.

Module 5: Functional Testing Deep Dive

1

Object Properties Testing

Verify UI element attributes including size, position, colour, font, visibility, enabled/disabled state, and default values match specifications.

2

Database Testing

Validate CRUD operations (Create, Read, Update, Delete), data integrity constraints, referential integrity, transaction handling, and data consistency across tables.

3

Error Handling Testing

Confirm appropriate error messages for invalid inputs, graceful degradation, exception handling, user-friendly messaging, and proper logging of errors.

4

Calculations Testing

Verify mathematical operations, formula accuracy, rounding behaviour, boundary values, and computational logic across various scenarios and edge cases.

Links Testing

- **Internal Links**: Navigate within application
- **External Links**: Open external websites
- **Anchor Links**: Jump to page sections
- **Email Links**: Launch email clients
- Verify correct URLs and targets
- Check broken link detection

Cookie & Session Testing

- Cookie creation and expiration
- Session persistence across pages
- Session timeout behaviour
- Cookie deletion and clearing
- Secure cookie attributes
- Session hijacking prevention

Module 6: Non-Functional Testing

Load Testing

Expected user load performance

Stress Testing

Breaking point identification

Scalability Testing

System growth capacity

Spike Testing

Sudden load increase handling

Volume Testing

Large data volume handling

Endurance Testing

Sustained load over time

Security Testing

- **Authentication**: Verify user identity mechanisms
- **Authorisation**: Validate access controls and permissions
- **Encryption**: Ensure data protection in transit and at rest
- SQL injection prevention
- Cross-site scripting (XSS) protection
- Session management security

Compatibility Testing

- **Hardware**: Different devices and configurations
- **Operating Systems**: Windows, macOS, Linux, mobile OS
- **Browsers**: Chrome, Firefox, Safari, Edge
- Screen resolutions and orientations
- Network conditions and bandwidth
- Third-party software integration

01

Fresh Installation

Clean install on new system

02

Upgrade Installation

Update from previous version

03

Uninstall Testing

Complete removal verification

04

Reinstall Testing

Install after uninstallation

Module 7: Testing Types & Techniques

Regression Testing

Re-test existing functionality after changes to ensure no new defects introduced. Types include unit regression, regional regression, and full regression testing.

Retesting

Verify specific defects are fixed. Focused on previously failed test cases, confirming bug resolution without broader scope.

Smoke Testing

Build Verification Testing (BVT) checks critical functionality before detailed testing. Quick validation that basic features work, determining if build is stable enough for further testing.

Sanity Testing

Narrow, focused testing after minor changes. Verifies specific functionality or bug fixes work correctly without comprehensive testing of entire application.

Exploratory Testing

Simultaneous learning, test design, and execution. Testers explore application without predefined scripts, using creativity and domain knowledge to find defects.

Ad-Hoc Testing

Informal, unstructured testing without planning or documentation. Random testing based on tester's intuition and experience, often revealing unexpected issues.

Monkey Testing

Random, chaotic input testing simulating unpredictable user behaviour. Provides invalid, unexpected data to assess system stability and error handling.

Positive vs Negative Testing

Positive Testing validates expected behaviour with valid inputs. **Negative Testing** ensures proper handling of invalid inputs, edge cases, and error conditions.

Globalisation vs Localisation

Globalisation Testing ensures application works across cultures and regions. **Localisation Testing** verifies adaptation to specific locales (language, currency, formats).

Module 8: Test Design Techniques

Equivalence Class Partitioning

Divide input domain into valid and invalid equivalence classes. Test one value from each class, reducing test cases whilst maintaining coverage. Example: Age field (0-17 invalid, 18-65 valid, 66+ invalid).

Boundary Value Analysis

Test values at boundaries between equivalence classes. Focus on minimum, maximum, just below minimum, and just above maximum values where defects commonly occur.

Decision Table Testing

Represent complex business logic with conditions, actions, and rules. Each column represents a test case combining different condition values, ensuring all combinations are tested.

State Transition Testing

Model system behaviour through states, events, and transitions. Test all valid transitions and verify invalid transitions are rejected.

- Identify all possible states
- Define triggering events
- Map valid transitions
- Test invalid transition handling

Error Guessing

Based on past failures, common mistakes, and domain expertise to identify likely problem areas.

- Analyse historical defects
- Consider common error patterns
- Apply domain knowledge
- Test intuitive failure points

Technique	Best For	Coverage	Effort
ECP	Large input domains	Moderate	Low
BVA	Numeric ranges	High at boundaries	Low
Decision Table	Complex logic	Complete combinations	Medium
State Transition	Sequential systems	All transitions	Medium
Error Guessing	Supplementary testing	Targeted	Low

Module 9: Test Documentation & Management

1. **Requirement Analysis** – Review and understand requirements, identify testable features
2. **Test Planning** – Define strategy, scope, resources, schedule, and deliverables
3. **Test Case Development** – Create detailed test cases, test data, and test scripts
4. **Environment Setup** – Configure test environments, tools, and test data
5. **Test Execution** – Run test cases, log defects, track results
6. **Test Closure** – Evaluate criteria, document metrics, lessons learnt

Test Case ID	Test Scenario	Test Steps	Expected Result	Status
TC_001	Valid login	1. Enter valid username 2. Enter valid password 3. Click Login	User logged in successfully, dashboard displayed	Pass
TC_002	Invalid password	1. Enter valid username 2. Enter invalid password 3. Click Login	Error message: "Invalid credentials"	Pass

The **Defect Life Cycle** tracks bugs from discovery through resolution: New → Assigned → Open → Fixed → Retest → Verified → Closed. Defects may be Rejected, Deferred, or Reopened based on validation and priority.

Module 10: Agile Testing & Tools

Sprint Planning

Select and commit to sprint backlog items

Development

Build features with continuous testing

Testing

Validate functionality and quality

Sprint Review

Demonstrate completed work

Retrospective

Reflect and improve processes

Testing in Agile

- Continuous testing throughout sprint
- Test-driven development (TDD)
- Behaviour-driven development (BDD)
- Automated regression testing
- Exploratory testing sessions
- Collaboration with developers
- Definition of Done includes testing

JIRA for Test Management

- Create and organise test cases
- Link tests to user stories
- Track test execution status
- Log and manage defects
- Generate test reports
- Sprint burndown charts
- Traceability and metrics

01

Version Control Basics

Git fundamentals: clone, commit, push, pull, branching

02

GitHub Collaboration

Repository management, pull requests, code reviews

03

Real Project Execution

Apply testing skills to actual project scenarios

04

Interview Preparation

Resume building, common questions, technical assessments

Selenium Fundamentals

10x
Faster Execution

90%
Cost Reduction

24/7
Continuous Testing

When to Automate

- Repetitive test cases
- Regression testing
- Data-driven scenarios
- Cross-browser testing
- Performance testing
- Stable functionality

When Manual Testing is Better

- Exploratory testing
- Usability testing
- Ad-hoc testing
- Frequently changing features
- One-time test scenarios
- Visual design verification

E2E Tests
Few, slow, expensive

Integration Tests
Moderate quantity and speed

Unit Tests
Many, fast, cheap

Locator	Syntax	Use Case
ID	driver.find_element(By.ID, "username")	Fastest, most reliable if unique
Name	driver.find_element(By.NAME, "email")	Good for form elements
CSS Selector	driver.find_element(By.CSS_SELECTOR, ".btn-primary")	Flexible, faster than XPath
XPath	driver.find_element(By.XPATH, "//button[@type='submit']")	Complex navigation, text-based
Link Text	driver.find_element(By.LINK_TEXT, "Sign Up")	Exact link text matching

Module 2: Advanced Selenium & Frameworks

Implicit Wait

Global wait applied to all elements. Polls DOM for specified time before throwing exception. Simple but less flexible.

Explicit Wait

Wait for specific conditions on particular elements. More control and reliability. Use WebDriverWait with expected conditions.

Fluent Wait

Custom polling interval and ignore specific exceptions. Maximum flexibility for complex scenarios with dynamic elements.

Handling Complex Elements

- **Alerts**: accept(), dismiss(), getText(), sendKeys()
- **Windows**: getWindowHandles(), switchTo().window()
- **iFrames**: switchTo().frame(), defaultContent()
- **Shadow DOM**: JavaScript executor for shadow roots
- **Dynamic Elements**: Explicit waits with custom conditions

Actions Class Operations

- **Mouse**: moveToElement(), clickAndHold(), dragAndDrop()
- **Context Menu**: contextClick() for right-click
- **Double Click**: doubleClick() for special interactions
- **Keyboard**: keyDown(), keyUp(), sendKeys()
- **Hover**: moveToElement() for dropdown menus

Data-Driven Framework

External data sources drive test execution

Keyword-Driven Framework

Keywords represent actions and test steps

Hybrid Framework

Combines data-driven and keyword-driven approaches

Page Object Model

Separate page elements from test logic

Page Object Model (POM) design pattern creates object repositories for web pages. Each page class contains locators and methods for that page, improving maintainability and reducing code duplication. Page Factory pattern further simplifies element initialisation using annotations.

Module 3: Playwright Modern Automation

Auto-Waiting

Intelligent waits eliminate flaky tests without explicit waits

Multi-Browser

Chromium, Firefox, WebKit support with single API

Network Control

Intercept, mock, and modify network requests

AI-Powered

Self-healing locators adapt to UI changes

Playwright vs Selenium

Feature	Playwright	Selenium
Speed	Faster	Slower
Auto-waiting	Built-in	Manual
Network mocking	Native	External tools
Debugging	Excellent	Basic
Mobile	Emulation	Appium needed

Smart Locators

- **getByRole**: Accessibility-first selection
- **getByText**: Find by visible text
- **getByLabel**: Form label association
- **getByTestId**: Stable test identifiers
- **Chaining**: Combine locators for precision
- **Auto-Retry**: Automatic retries on failure

Codegen

Record browser interactions and generate test code automatically. Accelerates test creation and provides learning examples for new users.

Playwright Inspector

Step through tests, inspect locators, and debug failures interactively. Essential tool for troubleshooting complex test scenarios.

Trace Viewer

Detailed execution traces with screenshots, network logs, and DOM snapshots. Powerful post-mortem debugging for CI/CD failures.

Module 4: AI-Powered Testing & Cloud Platforms

Self-Healing

Auto-fix broken locators

Test Generation

AI creates test cases

Visual Testing

Image comparison AI

Failure Analysis

AI diagnoses issues

Smart Maintenance

Reduce test upkeep

Headless vs Headed Execution

Headless Mode:

- Faster execution without GUI
- Lower resource consumption
- Ideal for CI/CD pipelines
- Docker container compatibility

Headed Mode:

- Visual debugging capability
- Development and troubleshooting
- Demo and presentation purposes

Cloud Testing Platforms

BrowserStack: 3000+ real devices and browsers

Sauce Labs: Comprehensive test orchestration

LambdaTest: Cost-effective parallel testing

Benefits:

- No infrastructure maintenance
- Instant scalability
- Real device testing
- Parallel execution
- Detailed analytics

Capture Baseline

Take reference screenshots of correct UI state

Run Tests

Execute tests and capture current screenshots

Compare Images

Pixel-by-pixel comparison identifies differences

Review Changes

Approve intentional changes or report bugs

Module 5: CI/CD Integration & Best Practices

Code Commit

Developer pushes code to repository

Build Trigger

CI/CD pipeline automatically starts

Run Tests

Automated tests execute in parallel

Generate Reports

Test results and artifacts created

Deploy

Successful builds deployed to environments

Test Design Principles

Independent, repeatable tests with clear assertions. Single responsibility per test. Avoid test interdependencies.

Code Reusability

Utility classes, helper methods, and Page Object Model. DRY principle reduces maintenance burden.

Avoiding Flakiness

Proper waits, stable locators, isolated test data. Retry mechanisms for transient failures. Deterministic test execution.

Test Data Management

Separate test data from code. Use data builders and factories. Clean up after tests. Environment-specific configurations.

Challenge	**Solution**
Flaky Tests	Implement proper waits, use stable locators, isolate test data, add retry logic
StaleElementReference	Re-locate elements, use explicit waits, avoid storing WebElement references
TimeoutException	Increase wait times, verify element visibility, check network conditions
Dynamic Locators	Use relative XPath, CSS selectors, or AI-powered self-healing locators
Slow Execution	Parallel execution, headless mode, optimise waits, reduce unnecessary steps

API Testing Fundamentals

10x	95%	Early
Faster Execution	Higher Coverage	Defect Detection
API tests run significantly faster than UI tests	Test business logic directly without UI dependencies	Find issues before UI development completes

GET

Retrieve data from server. Idempotent and safe. No request body. Parameters in URL.

POST

Create new resources. Not idempotent. Request body contains data. Returns created resource.

PUT

Update/replace entire resource. Idempotent. Request body contains complete data.

PATCH

Partial resource update. Request body contains only changed fields.

DELETE

Remove resources. Idempotent. May include request body for complex deletions.

HTTP Status Codes

- **2xx Success:** 200 OK, 201 Created, 204 No Content
- **3xx Redirection:** 301 Moved Permanently, 302 Found
- **4xx Client Errors:** 400 Bad Request, 401 Unauthorised, 403 Forbidden, 404 Not Found
- **5xx Server Errors:** 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable

Request Components

- **URL/URI:** Endpoint address and path
- **Headers:** Content-Type, Authorization, Accept
- **Query Parameters:** ?key=value&filter=active
- **Path Parameters:** /users/{id}/orders
- **Request Body:** JSON, XML, or form data

JSON (JavaScript Object Notation) is the standard data format for modern APIs. It uses key-value pairs, supports objects and arrays, and is human-readable. JSONPath expressions enable querying nested data structures for validation and extraction.

Module 2: Postman API Testing Tool

01

Install Postman

Download desktop app or use web version

02

Create Workspace

Organise APIs by project or team

03

Build Requests

Configure method, URL, headers, body

04

Write Tests

Add assertions and validations

05

Run Collections

Execute multiple requests sequentially

Variables in Postman

Type	Scope	Use Case
Global	All workspaces	Common values
Environment	Selected environment	Dev/QA/Prod URLs
Collection	Specific collection	Collection-specific data
Local	Single request	Temporary values
Dynamic	Runtime generation	$guid, $timestamp

API Key Authentication

Simple key-based authentication passed in headers or query parameters. Easy to implement but less secure.

Bearer Token

Token-based authentication using JWT or OAuth tokens in Authorization header. Industry standard for modern APIs.

OAuth 2.0

Delegated authorization framework. Secure, token-based access without sharing credentials. Complex but powerful.

Data-Driven Testing: Use CSV or JSON files to run the same request with different data sets. Collection Runner iterates through data, enabling comprehensive test coverage with minimal effort.

Module 3: API Automation with Python

Requests Library Basics

Pytest API Testing

Project Structure

Organise tests, utilities, config, and test data in logical folders

Configuration Management

Environment-specific settings in config files or environment variables

Utility Functions

Reusable helpers for authentication, data generation, and assertions

Test Data Management

External data files (JSON, CSV) for data-driven testing

Logging & Reporting

Comprehensive logs and HTML reports for test results

Session Objects persist parameters across requests, improving performance and simplifying authentication. Sessions maintain cookies, connection pooling, and default headers throughout multiple API calls.

Module 4: Advanced API Testing & GraphQL

API Chaining

Extract data from one response to use in subsequent requests

Database Validation

Verify API changes reflect correctly in database

Performance Testing

Measure response times, throughput, and scalability

OWASP API Security Top 10

1. Broken Object Level Authorization
2. Broken User Authentication
3. Excessive Data Exposure
4. Lack of Resources & Rate Limiting
5. Broken Function Level Authorization
6. Mass Assignment
7. Security Misconfiguration
8. Injection
9. Improper Assets Management
10. Insufficient Logging & Monitoring

GraphQL vs REST

Aspect	GraphQL	REST
Endpoints	Single endpoint	Multiple endpoints
Data fetching	Request exactly what you need	Fixed response structure
Over-fetching	Eliminated	Common issue
Versioning	Not required	Version in URL

GraphQL Queries

Fetch data with precise field selection. Reduces over-fetching and under-fetching.

GraphQL Mutations

Modify server-side data. Create, update, or delete operations.

GraphQL Subscriptions

Real-time data updates via WebSocket. Server pushes changes to clients.

Contract Testing ensures API providers and consumers agree on interface specifications. Pact framework enables consumer-driven contracts, catching integration issues early without requiring full integration environments.

Module 5: API Testing Best Practices & CI/CD

Folder Structure
Logical organisation of tests, utilities, configs, and data

Configuration Management
Environment-specific settings and credentials

Test Data Management
External data sources and data builders

Utility Classes
Reusable helpers and common functions

Logging & Reporting
Comprehensive logs and visual reports

1. **Code Commit** – Push API test changes to repository
2. **Trigger Pipeline** – GitHub Actions or Azure DevOps starts
3. **Run API Tests** – Execute test suite against environments
4. **Generate Reports** – Create Allure or HTML reports
5. **Notify Team** – Send results via email or Slack

Challenge	Solution
Authentication Issues	Use environment variables for credentials, implement token refresh logic
Data Dependencies	Create test data programmatically, use database seeding, implement cleanup
Environment Differences	Configuration files per environment, feature flags, environment detection
API Versioning	Test multiple versions, maintain backward compatibility tests
Response Time Variations	Set reasonable timeouts, implement retry logic, monitor performance trends

hello@digitaledify.ai +91 6304982304 Digital Edify

Python Fundamentals

1st
Most Popular
Top language for AI, testing, and data science

35
Keywords
Reserved words defining Python syntax and structure

Easy
Learning Curve
Readable syntax resembling natural language

Data Types

- **Simple:** int, float, bool, str
- **Complex:** list, tuple, dict, set
- **Type Conversion:** int(), float(), str(), bool()
- **Type Casting:** Explicit conversion between types

Operators

- **Arithmetic:** +, -, *, /, //, %, **
- **Comparison:** ==, !=, >, <, >=, <=
- **Logical:** and, or, not
- **Assignment:** =, +=, -=, *=, /=
- **Membership:** in, not in
- **Identity:** is, is not

if Statement
Execute code block if condition is True

elif Statement
Check additional conditions if previous conditions are False

else Statement
Execute code block if all conditions are False

match-case
Pattern matching for multiple conditions (Python 3.10+)

while Loop

Repeats whilst condition is True. Use break to exit early, continue to skip iteration.

for Loop

Iterates over sequences. range() generates number sequences efficiently.

Module 2: String Manipulation

String Indexing & Slicing

String Formatting

Case Conversion

upper(), lower(), capitalize(), title(), swapcase() methods transform string case for normalisation and comparison.

Search Methods

find(), index(), count() locate substrings. find() returns -1 if not found, index() raises exception.

Checking Methods

isalpha(), isdigit(), isalnum(), isspace(), isupper(), islower() validate string content and format.

Trimming & Replacement

Split & Join

String Immutability: Strings cannot be modified after creation. Operations like replace() or upper() return new strings rather than modifying the original. This ensures data integrity but requires awareness when processing large strings.

Module 3: Lists & Tuples

List Operations

List Methods

Lists and tuples store collections of data. This module covers list creation, indexing, slicing, operations, methods for adding (append, insert, extend), removing (remove, pop, clear), searching (index, count), sorting, reversing, list comprehensions, tuple creation, immutability, packing and unpacking, and comparing lists versus tuples.

1

Lists: Mutable & Dynamic

Lists can be modified after creation. Add, remove, or change elements. Use square brackets []. Slower than tuples but more flexible. Ideal for collections that change.

2

Tuples: Immutable & Fixed

Tuples cannot be modified after creation. Use parentheses (). Faster and more memory-efficient than lists. Ideal for fixed collections and function returns.

Tuple Operations

```
# Creating tuples
coordinates = (10, 20)
single = (42,)  # Note the comma

# Accessing elements
x = coordinates[0]
y = coordinates[1]

# Tuple packing
point = 10, 20, 30

# Tuple unpacking
x, y, z = point
```

When to Use Each

Use Lists When:	**Use Tuples When:**
Data changes frequently	Data is constant
Need to add/remove items	Returning multiple values
Order may change	Dictionary keys
Single type of data	Heterogeneous data

Module 4: Dictionaries & Sets

Dictionaries and sets provide powerful data structures. This module covers dictionary creation, key-value access, operations, methods (keys, values, items), dictionary comprehensions, nested dictionaries, set creation, properties (unique, unordered, unchangeable), set operations (union, intersection, difference), mathematical set operations, subset and superset checks, and frozen sets.

Dictionary Operations

Dictionary Methods

Unique Elements

Sets automatically remove duplicates

Unordered

No guaranteed order of elements

Unchangeable Items

Elements cannot be modified, only added/removed

Set Operations

Set Methods

Frozen Sets are immutable versions of sets. Created with frozenset(), they can be used as dictionary keys or elements of other sets. Useful when you need set operations but require immutability.

Module 5: Advanced Collections & Iterators

Advanced Python collections enhance functionality. This module explores the collections module (namedtuple, Counter, defaultdict, deque), iterators and iteration protocol, custom iterators, generators with yield statement, generator expressions, memory efficiency concepts, lambda functions, higher-order functions (map, filter, reduce), functional programming, and generator pipelines.

namedtuple

Tuple subclass with named fields. Readable and self-documenting. Immutable like tuples but accessible by name.

Counter

Dict subclass for counting hashable objects. Convenient for frequency analysis and statistics.

defaultdict

Dict with default values for missing keys. Eliminates KeyError exceptions and simplifies code.

deque

Double-ended queue with fast appends and pops from both ends. Efficient for queues and stacks.

Generators

Generators produce values on-demand, saving memory for large datasets.

Lambda & Higher-Order Functions

Lambda functions are concise, anonymous functions for simple operstions.

1

Iterator Protocol

Objects implementing __iter__() and __next__()

2

Custom Iterators

Create classes with iteration behaviour

3

Generator Functions

Use yield to create iterators simply

4

Generator Pipelines

Chain generators for efficient data processing

Module 6: Functions & Scope

Functions enable code reusability and organisation. This module covers function definition and calling, parameters and arguments (positional, keyword, default), arbitrary arguments (*args, **kwargs), return statements, multiple return values, local and global scope, global keyword, built-in and user-defined functions, lambda functions, docstrings, and recursive functions.

Function Basics

A Function is reusable block of code that performs a specific task and runs when called.

Argument Types

Functions accept inputs as positional, keyboard, default, or variable-length arguments to handle different use cases.

Local Scope

Variables defined inside functions. Only accessible within that function. Created when function is called, destroyed when function returns.

Global Scope

Variables defined outside functions. Accessible throughout the module. Use global keyword to modify global variables inside functions.

Enclosing Scope

Variables in outer functions accessible to nested functions. Enables closures and function factories. Use nonlocal keyword to modify.

Lambda Functions

Short, anonymous functions written in a single line, mainly used for simple operations and functional programming.

Recursive Functions

Functions that call themselves to solve a problem by breaking it into smaller, repeatable steps.

Module 7: Modules & Packages

Built-in Modules

Standard library modules included with Python

User-Defined Modules

Custom modules created for specific projects

External Packages

Third-party modules installed via pip

Importing Techniques

Different ways to access modules in Python, such as importing entire modules or specific components for better code organisation.

Common Built-in Modules

- **math:** Mathematical functions and constants
- **random:** Random number generation
- **datetime:** Date and time manipulation
- **os:** Operating system interface
- **sys:** System-specific parameters
- **json:** JSON encoding and decoding
- **re:** Regular expressions

Create Package Directory

Folder containing __init__.py and module files

Add __init__.py

Marks directory as Python package, can contain initialisation code

Create Module Files

Python files (.py) containing functions and classes

pip Package Manager

A tool used to install, update, and manage external python libraries required for development

Popular External Packages

- **requests:** HTTP library for API calls
- **pandas:** Data analysis and manipulation
- **numpy:** Numerical computing
- **pytest:** Testing framework
- **selenium:** Web automation
- **beautifulsoup4:** Web scraping
- **flask:** Web framework

Module 8: Working with Data Formats

File operations and data formats are essential for testing. This module covers file operations basics (CRUD), open() function and modes, reading files (read, readline, readlines), writing and appending, file paths, directory management (os, shutil), CSV file operations (reader, writer, DictReader, DictWriter), JSON operations (dump, dumps, load, loads), and data serialization.

File Operations

Reading, Writing, and Updating files to tore and manage data efficiently.

File Modes

Mode	Description
'r'	Read (default)
'w'	Write (overwrites)
'a'	Append
'r+'	Read and write
'rb'	Read binary
'wb'	Write binary

CSV Reader

CSV Writer

CSV DictReader

JSON Operations

Handling structured data by converting between JSON format and Python objects

Directory Management

Creating, navigating, and organizing folders and files using system utilities.

Module 9: Advanced Python Concepts

Exception Handling

Managing runtime errors gracefully using try-except block to prevent program crashes

Custom Exceptions

Creating user-defined error types to handle specific application scenarios clearly.

Function Decorators

Modify or enhance function behaviour without changing function code. Wrap functions with additional functionality.

Timing Decorator

Measure execution time of functions. Useful for performance testing and optimization.

Logging Decorator

Automatically log function calls, arguments, and return values for debugging and monitoring.

Retry Decorator

Automatically retry failed operations with configurable attempts and delays.

Decorator

Enhancing function behaviour dynamically without modifying the original code

Context Managers

Managing resources safely using automatic setup and cleanup mechanisms

Object-Oriented Programming

SECTION 5: PYTHON FOR AI & TESTING

MODULE 10

OOP enables modular, maintainable code. This module covers OOP fundamentals, classes and objects, attributes (instance and class variables), __init__ constructor, self parameter, instance methods, class methods, static methods, and the four pillars: Encapsulation (access modifiers), Inheritance (single, multi-level, multiple), Abstraction (abstract classes and methods), and Polymorphism (method overriding, duck typing).

Classes & Objects

Blueprints and Instances used to structure data and behaviour in programs

Encapsulation

Protecting data by controlling access through methods and access levels

Encapsulation

Bundle data and methods, control access

Inheritance

Derive classes from existing classes

Polymorphism

Same interface, different implementations

Abstraction

Hide complexity, show essential features

Inheritance

Reusing and extending existing classes to build hierarchical relationships

Abstraction & Polymorphism

Simplifying complex systems and enabling different behaviours through a common interfaces

PostgreSQL Foundations

Atomicity

All operations in a transaction succeed or all fail. No partial transactions.

Consistency

Database remains in valid state before and after transactions.

Isolation

Concurrent transactions don't interfere with each other.

Durability

Committed transactions persist even after system failures.

Data Types

Category	Types
Numeric	INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION
Character	CHAR, VARCHAR, TEXT
Date/Time	DATE, TIME, TIMESTAMP, INTERVAL
Boolean	BOOLEAN (TRUE/FALSE)
Special	JSON, JSONB, ARRAY, UUID

Constraints

- **PRIMARY KEY:** Unique identifier for rows
- **FOREIGN KEY:** References another table's primary key
- **UNIQUE:** Ensures column values are unique
- **NOT NULL:** Prevents null values
- **CHECK:** Validates data against conditions
- **DEFAULT:** Provides default values

Creating Tables

Creating tables is the process of defining the structure of a database table by specifying column names, data types, and rules for storing data.

Inserting Data

Inserting data is the process of adding new records (rows) into a database table.

Module 2: Querying & Analyzing Data

Querying retrieves and analyzes data. This module covers SELECT statement basics, column aliases, WHERE clause filtering, comparison and logical operators, BETWEEN, IN, LIKE operators, NULL handling, ORDER BY sorting, DISTINCT, LIMIT and OFFSET pagination, string/numeric/date functions, aggregate functions (COUNT, SUM, AVG, MIN, MAX), GROUP BY, HAVING, window functions, and JOIN operations.

SELECT & Filtering

Select retrieves specific data from a table, while filtering restricts the results based on given conditions.

Sorting & Limiting

Sorting arranges query results in a specific order, and limiting controls the number of records displayed.

String Functions

UPPER(), LOWER(), CONCAT(), SUBSTRING(), LENGTH(), TRIM(), REPLACE() manipulate text data.

Numeric Functions

ROUND(), CEIL(), FLOOR(), ABS(), POWER(), SQRT() perform mathematical operations.

Date Functions

CURRENT_DATE, EXTRACT(), DATE_TRUNC(), AGE() handle date and time calculations.

Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single summarized result.

JOIN Operations

Join operations combine data from two or more tables based on a related column.

Module 3: Advanced Queries & Data Manipulation

Advanced SQL techniques handle complex scenarios. This module covers subqueries in WHERE, SELECT, and FROM clauses, correlated subqueries, EXISTS and NOT EXISTS, IN and NOT IN with subqueries, Common Table Expressions (CTEs), recursive CTEs for hierarchical data, set operators (UNION, INTERSECT, EXCEPT), UPDATE statements, DELETE operations, TRUNCATE, and transaction management.

Subqueries

A subquery is a query nested inside another query to supply data to the main query.

Common Table Expressions

A Common Table Expression is a temporary named result set that improves query readability and organization.

UNION Combine results, remove duplicates	UNION ALL Combine results, keep duplicates
INTERSECT Return common rows from both queries	EXCEPT Return rows in first query but not second

UPDATE & DELETE

Update modifies existing data in a table, while Delete removes records from a table.

Transaction Management

Transaction management controls a sequence of database operations to ensure data consistency, integrity, and reliability.

Module 4: Database Programming & Automation

ALTER TABLE & Indexes

ALTER TABLE modifies the structure of an existing table, while indexes improve data retrieval speed by optimizing searches.

Views & Materialized Views

A view is a virtual table based on a query, whereas a materialized view stores the query result physically for faster access.

Stored Functions

Reusable SQL logic returning values. Can be called in queries and expressions.

Stored Procedures

Execute complex operations without returning values. Support transaction control.

Triggers

Automatically execute functions in response to data changes (INSERT, UPDATE, DELETE).

Audit Logging

Track data changes automatically using triggers for compliance and debugging.

PL/pgSQL Function

A PL/pgSQL function is a reusable block of procedural code in PostgreSQL used to perform complex operations and return results.

Trigger

A trigger automatically executes an action when a specified event occurs on a table, such as updating a log when a record is inserted.

Module 5: Database Design & Optimization

Entities

Objects or concepts with data

Attributes

Properties describing entities

Cardinality

1:1, 1:M, M:N relationships

Relationships

Connections between entities

First Normal Form (1NF)

Eliminate repeating groups, atomic values in columns

Second Normal Form (2NF)

Meet 1NF, remove partial dependencies on composite keys

Third Normal Form (3NF)

Meet 2NF, remove transitive dependencies

Performance Tuning

Indexes, Statistics, VACUUM, Connection Pooling, Partitioning, Caching, Query Rewriting

Best Practice	**Description**
Naming Conventions	Use lowercase with underscores (snake_case), descriptive names, plural for tables
Data Type Selection	Choose appropriate types, avoid VARCHAR(MAX), use TIMESTAMP for dates
Primary Keys	Use SERIAL or UUID, ensure uniqueness, index automatically created
Foreign Keys	Enforce referential integrity, index foreign key columns for joins
Normalization	Normalize to 3NF, denormalize strategically for performance

SECTION 7: GENERATIVE AI & AGENTIC AI MODULE 1

Foundations of Generative AI

GPT-1 (2018)

117M parameters, unsupervised pre-training

GPT-2 (2019)

1.5B parameters, improved coherence

GPT-3 (2020)

175B parameters, few-shot learning

GPT-4 (2023)

Multimodal, advanced reasoning

2026 Models

Frontier models with enhanced capabilities

Transformer Architecture

- **Self-Attention, Multi-Head Attention, Positional Encoding, Feed-Forward Networks, Layer Normalization, Residual Connections**

Tokenization

Breaking text into tokens (words, subwords, characters) for model processing. Different models use different tokenization strategies: **Byte-Pair Encoding (BPE), WordPiece, SentencePiece.**

Model	Provider	Strengths	Context Window	Best For
GPT-4	OpenAI	Reasoning, coding	128K tokens	Complex tasks
Claude 3	Anthropic	Safety, analysis	200K tokens	Long documents
Gemini	Google	Multimodal, search	1M tokens	Research, integration
DeepSeek	DeepSeek	Cost-effective	64K tokens	Budget-conscious

Modules 2, 3, & 4: Prompt Engineering & RAG

Effective prompts and retrieval systems maximize LLM capabilities. This module covers advanced prompt engineering, context design, reasoning mode optimization, reducing hallucinations, zero-shot and few-shot prompting, chain-of-thought, multimodal prompting, LLM APIs (OpenAI, Anthropic, Google, DeepSeek), LangChain 1.0, vector databases (ChromaDB, Pinecone, Qdrant), and production RAG pipelines.

Clear Instructions

Provide explicit, detailed instructions with context and constraints

Few-Shot Examples

Include examples demonstrating desired output format and style

Chain-of-Thought

Request step-by-step reasoning for complex problems

Role Assignment

Define model's role and expertise for better responses

Reducing Hallucinations

- Provide relevant context and facts
- Request citations and sources
- Use retrieval-augmented generation (RAG)
- Ask model to acknowledge uncertainty
- Verify outputs against ground truth
- Use temperature parameter wisely
- Implement fact-checking pipelines

LangChain 1.0 Features

- **create_agent:** Simplified agent creation
- **Middleware:** Customization and control
- **Multi-Provider:** Support for multiple LLMs
- **Streaming:** Real-time response generation
- **Function Calling:** Structured outputs
- **Cost Optimization:** Efficient token usage

Document Processing

Load and chunk documents into manageable pieces

Embedding Generation

Convert text chunks into vector representations

Vector Storage

Store embeddings in vector database

Semantic Search

Retrieve relevant chunks for queries

Response Generation

LLM generates answer using retrieved context

ChromaDB

Open-source, embedded vector database. Easy setup, Python-native, ideal for prototyping and small-scale applications.

Pinecone

Managed vector database service. Scalable, fast, enterprise-ready with advanced filtering and hybrid search.

Qdrant

High-performance vector search engine. Self-hosted or cloud, supports filtering, and offers excellent performance.

Agentic RAG enhances traditional RAG with self-improving retrieval. Agents analyze queries, select optimal retrieval strategies, evaluate result quality, and iteratively refine searches for better accuracy and relevance.

Modules 5 & 6: Production Deployment & Agentic AI

Streamlit

Python framework for rapid AI app development. Simple syntax, interactive widgets, automatic reloading. Ideal for prototypes and internal tools.

Gradio

Create ML interfaces with minimal code. Supports various input/output types, shareable links, and easy deployment. Perfect for demos and testing.

LangGraph Platform

Production-grade deployment for LangGraph applications. Managed infrastructure, scaling, monitoring, and enterprise features.

Cost Optimization Strategies

Use the right model size for each task, optimize prompts, and batch or cache repeated queries. Stream long responses, monitor token usage, and leverage fine-tuned models for specific domains.

AI Governance & Compliance

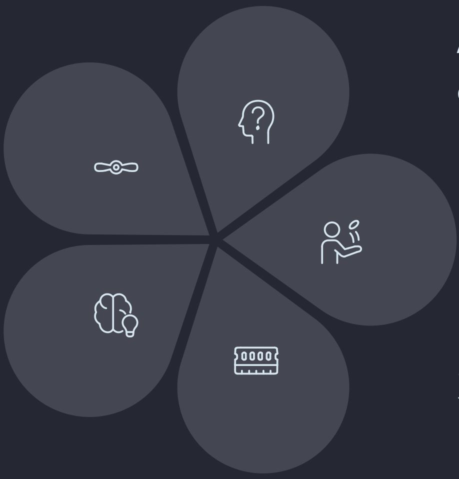
The EU AI Act enforces risk-based AI regulation, ensuring GDPR-compliant data use, transparent and explainable decisions, and fair outputs.

Planning

Break down complex tasks into steps

Learning

Improve from feedback and experience



Reasoning

Analyze information and make decisions

Action

Execute tasks using tools and APIs

Memory

Maintain context across interactions

Monitoring & Observability

Monitor API latency, response times, and token usage while logging all requests and responses. Track output quality, alert on errors or anomalies, and analyze user feedback.

Enterprise Integration

Enable secure access with SSO and RBAC, integrate with tools like Slack or Teams, and use API gateways for rate limiting.

Modules 7 & 8: LangGraph Workflows & Patterns

State Management

Maintain workflow state across nodes

Node Execution

Process data at each workflow step

Conditional Routing

Direct flow based on conditions

Parallel Processing

Execute multiple nodes simultaneously

Output Generation

Produce final workflow results

LangGraph 1.0 Features

Node Caching, Pre/Post Hooks, Type Safety, Streaming, Parallel Execution, Conditional Routine, Iterative Loops

Advanced Workflow Patterns

Decision Trees, Refinement Loops, Quality Gates, Multi-Stage Approval, Parallel Analysis, Error Recovery, Human-in-the-Loop

1 **Essay Submission**
Student submits essay for evaluation

2 **Initial Analysis**
LLM analyzes grammar, structure, content

3 **Quality Check**
Conditional routing based on quality score

4 **Refinement Loop**
Iterative improvement if score below threshold

5 **Final Evaluation**
Generate detailed feedback and grade

Customer Feedback Routing: LangGraph workflows analyze customer feedback sentiment, classify issues by category, route to appropriate departments, prioritize based on urgency, and track resolution status—all automated with conditional logic and parallel processing.

Modules 9, & 10: Production Agentic Systems

Durable State Management

Persist workflow state across sessions and failures. Enable long-running processes, resume after interruptions, and maintain context for multi-day workflows. PostgreSQL and Redis provide reliable storage.

Human-in-the-Loop (HITL)

Integrate human decision points in automated workflows. Pause for approval, gather feedback, handle exceptions, and ensure compliance. Essential for high-stakes decisions and regulatory requirements.

Multi-Agent System Design

Specialized Agents, Coordination, Communication, Task Decomposition, Result Aggregation, Conflict Resolution

Google A2A Protocol

Standardize agent communication for seamless interoperability, secure access, capability discovery, task delegation, result sharing, and reliable error recovery.

LangSmith Observability

Comprehensive monitoring and debugging. Trace every LLM call, visualize workflow execution, analyze costs, identify bottlenecks, and optimize performance.

MCP Security Model

Model Context Protocol security ensures safe tool access. Sandboxed execution, permission management, audit logging, and threat detection protect systems.

Prompt Injection Prevention

Defend against malicious prompts attempting to manipulate agent behaviour. Input validation, output filtering, and context isolation prevent attacks.

Agent Guardrails & Safety

Ensure safe and reliable operation through input validation, content filtering, rate limits, sensitive data detection, bias mitigation.

Compliance & Audit Trails

Track all agent actions, data changes, and version history, while generating compliance reports, supporting audits.

99.9%	<100ms	100%
Uptime	Latency	Audit Coverage