ocac

Technical Directorate of Information Technology Department

Govt. of Odisha

OCAC Training Centre

INTERNSHIP TRAINING PROGRAM

PYTHON

About OCAC Training Centre

OCAC Training Centre is the state-of-the-art training centre established by OCAC the Designated Technical Directorate of Information Technology Department, Govt. of Odisha to develop the IT skills of learners and make them job-ready/ workforce ready with support from Odisha Knowledge Corporation Ltd. (OKCL).

OCAC Training Centre provide Summer Training & Internship for all the technical Graduates in the state of Odisha.

Here are some key benefits of the program that we believe will be of interest to students:

- ✓ **Learn from expert trainers with real-world experience in C++ / Python / Java.**
- ✓ **Work on a short-term project that showcases their technical skills.**
- ✓ **Get personalized guidance and feedback to help them excel.**
- ✓ **Receive a completion certificate from OCAC, a reputed government organization.**
- ✓ **Develop essential interview skills to ace technical and personal interviews.**
- ✓ **Guidance & opportunity for Placement**

Content

Introduction

The OCAC Training Centre offers Internship Training programs to B-Tech and MCA students during the summer. These internship programs provide students with an opportunity to learn a programming language through assignment-based learning systems. The training program is designed to enhance students' practical skills and knowledge in their chosen programming language.

Here are some key features of the Internship Training program:

1. Programming Language Learning: Students will undergo classroom learning sessions where they will be taught the fundamentals and advanced concepts of a specific programming language. The focus is on developing a strong understanding of the language and its syntax.
2. Assignment-Based Learning Systems: The training program utilizes assignment-based learning systems, which means students will receive assignments related to the topics covered in the classroom. These assignments are designed to reinforce the theoretical concepts learned and encourage practical application.
3. Small Project Development: As part of the internship training, students will be required to work on a small project based on the learning outcomes. This project allows students to apply their knowledge and skills in a real-world scenario, fostering practical experience and problem-solving abilities.
4. Duration: The total duration of the Internship Training program is 60 hours. This includes 40 hours of classroom learning, where students receive theoretical instruction and hands-on practice, and an additional 20 hours dedicated to project development.

The Internship Training program aims to provide students with a comprehensive learning experience that combines theoretical knowledge with practical application. By completing assignments and working on a small project, students gain valuable skills and experience in their chosen programming language, preparing them for future career opportunities in the field of software development.

Course Curriculum for Internship Training in Python:

1. Brush-up Session on Core Python (10 hours):
    - Introduction to Python programming language
    - Variables, data types, and operators
    - Control flow statements: if-else, loops, and conditional statements
    - Functions and modules
    - File handling and I/O operations
    - Exception handling
    - Object-oriented programming (OOP) concepts: classes, objects, inheritance, and polymorphism
    - Working with lists, tuples, dictionaries, and sets
    - String manipulation and regular expressions
    - Introduction to basic data structures and algorithms

Assignment:

I. Introduction to Python programming language: a. Write a Python program to display the Fibonacci series up to a given number. b. Implement a program that calculates the factorial of a number using recursion. c. Design a program that checks whether a given string is a palindrome or not.

II. Variables, data types, and operators: a. Write a program that swaps the values of two variables without using a temporary variable. b. Design a program that calculates the area and perimeter of different shapes based on user input. c. Implement a program that converts temperature between Celsius and Fahrenheit.

III. Control flow statements: if-else, loops, and conditional statements: a. Create a program that prints all prime numbers in a given range using a loop and conditional statements. b. Design a program that finds the sum of all even numbers in a list using loop control statements. c. Implement a program that generates the Fibonacci series using a while loop.

IV. Functions and modules: a. Write a Python function that checks whether a given number is prime or not. b. Design a program that calculates the factorial of a number using a user-defined function. c. Implement a program that finds the largest and smallest elements in a list using built-in functions and modules.

V. File handling and I/O operations: a. Create a program that reads data from a text file, performs some operations, and writes the result to another file. b. Design a program that counts the frequency of words in a text file and displays the result. c. Implement a program that reads a CSV file and performs data analysis or manipulation.

VI. Exception handling: a. Write a program that handles exceptions when dividing two numbers and displays an appropriate error message. b. Design a program that reads data from a file, handles file-related exceptions, and displays error messages accordingly. c. Implement a program that prompts the user for input until a valid integer is entered, handling exceptions for incorrect input.

VII. Object-oriented programming (OOP) concepts: classes, objects, inheritance, and polymorphism: a. Create a class that represents a student and implement methods to calculate the average grade and display student information. b. Design a program that demonstrates inheritance by creating subclasses for different types of vehicles. c. Implement a program that showcases polymorphism by creating different shapes and calculating their areas.

VIII. Working with lists, tuples, dictionaries, and sets: a. Write a program that removes duplicates from a list using sets. b. Design a program that sorts a list of dictionaries based on a specific key. c. Implement a program that performs various operations on tuples like concatenation, slicing, and indexing.

IX. String manipulation and regular expressions: a. Create a program that checks whether a given string is a palindrome using string manipulation techniques. b. Design a program that replaces specific words in a text using regular expressions. c. Implement a program that validates email addresses using regular expressions.

X. Introduction to basic data structures and algorithms: a. Write a program that implements a stack or queue data structure and performs push, pop, and peek operations. b. Design a program that searches for a specific element in a list using linear search or binary search algorithms. c. Implement a program that sorts a list of numbers using different sorting algorithms like bubble sort or insertion sort.

2. Advanced Python (40 hours):

- Python Libraries and Packages:
    - NumPy: Introduction to numerical computing with arrays and mathematical operations
    - Pandas: Data manipulation and analysis using DataFrames
    - Matplotlib and Seaborn: Data visualization and plotting
    - BeautifulSoup: Web scraping and parsing HTML/XML data
    - Requests: HTTP requests and working with APIs
    - SQLAlchemy: Database connectivity and ORM (Object-Relational Mapping)
    - TensorFlow or PyTorch: Introduction to machine learning and deep learning frameworks

- Advanced Concepts:
    - Decorators and generators
    - Context managers and file handling best practices
    - Concurrency and parallelism: multithreading and multiprocessing
    - Regular expressions: advanced usage and pattern matching
    - Advanced data structures: stacks, queues, linked lists, and trees
    - Testing and debugging: unit testing frameworks and debugging techniques
    - Introduction to data science concepts: data preprocessing, feature extraction, and model evaluation

Assignment:

i. Python Libraries and Packages: a. Use NumPy to create a program that performs matrix multiplication and calculates the determinant of a matrix. b. Design a program that utilizes Pandas to read a CSV file, manipulate the data, and perform statistical analysis. c. Implement a program that uses Matplotlib or Seaborn to visualize data from a given dataset using different plot types.
ii. BeautifulSoup: a. Create a program that extracts specific information from a website by scraping its HTML content using BeautifulSoup. b. Design a program that parses an XML document using BeautifulSoup and retrieves data from specific tags.
iii. Requests: a. Write a program that sends HTTP requests to a RESTful API and retrieves data in JSON format, then processes and displays it. b. Implement a program that performs web scraping by making requests to multiple web pages and extracts relevant information.
iv. SQLAlchemy: a. Design a program that establishes a connection to a database using SQLAlchemy and performs CRUD (Create, Read, Update, Delete) operations. b. Create a program that utilizes ORM (Object-Relational Mapping) to define and interact with database tables and relationships.
v. TensorFlow or PyTorch: a. Implement a program that trains a basic neural network using TensorFlow or PyTorch to classify images from a given dataset. b. Design a program that utilizes a pre-trained model from TensorFlow or PyTorch for image recognition and displays the results.
vi. Decorators and generators: a. Write a program that uses decorators to measure the execution time of a function and log the results. b. Implement a program that uses generators to generate and iterate through a sequence of prime numbers.
vii. Context managers and file handling best practices: a. Create a program that uses context managers to handle file operations and ensure proper resource management. b. Design a program that reads data from multiple files, combines and processes the data, and writes the result to a new file using context managers.

viii. Concurrency and parallelism: multithreading and multiprocessing: a. Write a program that demonstrates multithreading to perform concurrent tasks and measure the speedup in execution time. b. Design a program that utilizes multiprocessing to process multiple tasks in parallel and compare the performance with sequential execution.

ix. Regular expressions: advanced usage and pattern matching: a. Create a program that uses regular expressions to extract email addresses or phone numbers from a given text. b. Implement a program that validates and filters URLs based on a specific pattern using regular expressions.

x. Advanced data structures: a. Design a program that implements a stack data structure and evaluates a postfix expression. b. Write a program that utilizes a binary tree data structure to perform various operations like insertion, deletion, and traversal.

3. Project Work using Python (20 hours): Students will work on a hands-on project to apply their knowledge and skills acquired during the training. The project work will involve developing a practical application or solving a real-world problem using Python. The project will require students to demonstrate their understanding of core and advanced Python concepts, as well as their ability to design and implement Python-based solutions.

Syllabus for Brush-up Session on Core Python

Detailed Syllabus for Brush-up Session on Core Python (10 hours):

1. Introduction to Python programming language
    - Overview of Python and its key features
    - Setting up Python environment and running Python scripts
    - Understanding Python's syntax and indentation rules
    - Exploring the Python interactive shell and Integrated Development Environments (IDEs)
2. Variables, Data Types, and Operators
    - Declaring and using variables in Python
    - Data types: integers, floats, strings, booleans, and others
    - Type conversion and type checking
    - Arithmetic, comparison, logical, and assignment operators
    - Operator precedence and associativity
3. Control Flow Statements: if-else, loops, and conditional statements
    - Conditional statements: if, else, and elif
    - Comparison operators and logical operators in conditionals
    - Looping structures: for and while loops
    - Loop control statements: break, continue, and pass
    - Nested loops and loop optimization techniques
4. Functions and Modules
    - Defining and calling functions in Python
    - Function parameters and return values
    - Scope of variables: global and local variables
    - Built-in functions vs. user-defined functions
    - Modules and importing modules in Python
    - Exploring standard library modules and external libraries
5. File Handling and I/O Operations
    - Opening, reading, writing, and closing files in Python

- Different file modes: text files vs. binary files
- File object methods for file manipulation
- Handling exceptions and error checking during file operations
- Working with file paths and directories
- Standard input and output: reading from and writing to the console

6. Exception Handling
    - Understanding exceptions and error handling in Python
    - Exception handling using try-except blocks
    - Handling specific exceptions and multiple exceptions
    - Raising exceptions and creating custom exceptions
    - Using finally block and cleaning up resources
    - Exception propagation and traceback information
7. Object-Oriented Programming (OOP) Concepts
    - Introduction to OOP and its principles
    - Classes and objects: defining and using classes
    - Constructors and destructors in classes
    - Encapsulation: data hiding and access modifiers
    - Inheritance: creating derived classes and base classes
    - Polymorphism: method overriding and method overloading
8. Working with Lists, Tuples, Dictionaries, and Sets
    - Lists: creating, indexing, slicing, and modifying lists
    - Tuples: creating, accessing, and manipulating tuples
    - Dictionaries: creating, accessing, and updating key-value pairs
    - Sets: creating, adding, removing, and performing set operations
    - List comprehensions and other techniques for data manipulation
9. String Manipulation and Regular Expressions
    - String operations: concatenation, indexing, slicing, and formatting
    - String methods for manipulation and transformation
    - Regular expressions: pattern matching and search operations

- Regular expression metacharacters and character classes
- Using regular expressions for data validation and extraction
- Working with string functions and string formatting

10. Introduction to Basic Data Structures and Algorithms
    - Arrays and their manipulation in Python
    - Searching and sorting algorithms: linear search, binary search, and various sorting techniques
    - Introduction to stack, queue, linked list, and basic algorithms using these data structures
    - Introduction to recursion and recursive algorithms
    - Understanding time and space complexity of algorithms

Syllabus for Python Advanced

Detailed Syllabus for Python Advanced Concepts (40 hours):

1. Decorators and Generators
    - Understanding the concept of decorators and their role in Python
    - Creating and using decorators to modify the behavior of functions
    - Decorator chaining and applying multiple decorators
    - Exploring built-in decorators and creating custom decorators
    - Introduction to generators and their use in creating iterators
    - Implementing generator functions and using generator expressions
2. Context Managers and File Handling Best Practices
    - Working with context managers and the **with** statement in Python
    - Creating context managers using the contextlib module
    - Implementing context managers as classes and using the **__enter__** and **__exit__** methods
    - Handling file operations using context managers for clean and safe file handling
    - Understanding best practices for file handling, including error handling and resource management
3. Concurrency and Parallelism: Multithreading and Multiprocessing
    - Introduction to concurrent programming in Python
    - Working with threads and the threading module
    - Synchronization and thread safety using locks, semaphores, and condition variables
    - Understanding the Global Interpreter Lock (GIL) and its impact on multithreading
    - Parallel processing using multiple processes and the multiprocessing module
    - Interprocess communication and data sharing techniques
4. Regular Expressions: Advanced Usage and Pattern Matching
    - Advanced regular expression patterns and metacharacters
    - Lookahead and lookbehind assertions in regular expressions
    - Grouping, capturing, and backreferences in pattern matching

- Using regular expressions for advanced text manipulation and data extraction
- Regular expression performance optimization techniques
- Exploring the re module and its functions for pattern matching in Python

5. Advanced Data Structures: Stacks, Queues, Linked Lists, and Trees
    - Implementing and using stacks, queues, linked lists, and trees in Python
    - Understanding the characteristics, operations, and applications of these data structures
    - Implementing common algorithms on these data structures, such as traversal and search
    - Analyzing time and space complexity of operations on advanced data structures
    - Exploring advanced data structure concepts, such as balanced trees and graph algorithms
6. Testing and Debugging: Unit Testing Frameworks and Debugging Techniques
    - Introduction to software testing and the importance of unit testing
    - Exploring unit testing frameworks in Python, such as unittest and pytest
    - Writing test cases, test suites, and test fixtures
    - Test-driven development (TDD) approach and its benefits
    - Debugging techniques using breakpoints, logging, and exception handling
    - Profiling and performance optimization techniques
7. Introduction to Data Science Concepts: Data Preprocessing, Feature Extraction, and Model Evaluation
    - Overview of data science and its applications
    - Data preprocessing techniques: cleaning, normalization, and handling missing values
    - Feature extraction and feature engineering from raw data
    - Introduction to machine learning algorithms and model evaluation
    - Evaluating classification and regression models using metrics such as accuracy, precision, recall, and F1-score
    - Introduction to popular Python libraries for data science, such as NumPy, Pandas, and scikit-learn

Projects Using Python

Here's a table that provides details of the use-cases and technologies/tools required for each project, implemented using Python:

Project	Module-wise Use-Cases	Technologies/Tools/APIs
Online Quiz Application	User Authentication: Allow users to create accounts, log in, and manage their profiles	Flask, HTML/CSS, JavaScript
	Quiz Selection: Provide a list of available quizzes and allow users to select the desired quiz	Python
	Quiz Engine: Display quiz questions, validate answers, calculate scores, and provide instant feedback	Flask, Python
	Leaderboard: Maintain a leaderboard to track and display the highest scores and rankings among participants	SQLite, SQLAlchemy
File Transfer Application	Client Interface: Provide a user-friendly interface for clients to select and transfer files to the server	Python sockets
	Server Interface: Receive files from clients, store them, and notify clients about the successful file transfer	Python sockets
Social Media Analytics Tool	Data Retrieval: Fetch social media data using APIs and retrieve information such as user engagement, post popularity, and sentiment analysis	Twitter/Facebook APIs, Python requests

	Data Analysis: Analyze the retrieved data to identify trends, popular topics, user sentiment, and engagement metrics	Python libraries for data analysis
	Visualization: Present the analyzed data using interactive charts, graphs, and visual representations	Matplotlib, Plotly
Employee Management System	Employee Records: Manage employee information, including adding new employees, updating details, and searching for employee records	Django, SQLite
	Attendance Management: Track employee attendance, leave records, and generate attendance reports	Django, HTML/CSS
Weather Forecast Application	Data Retrieval: Fetch weather data from a weather API and retrieve information such as current weather conditions and forecasts	Weather API, Python requests
	User Interface: Display weather information in a user-friendly format, including current weather, temperature, humidity, and forecasts	Flask, HTML/CSS
Online Voting System	Voting Engine: Enable users to cast votes for different categories, count votes, and determine the winners	Flask, Python

	User Authentication: Implement user registration, login, and profile management functionalities	Flask, HTML/CSS
	Real-time Voting Results: Display real-time voting results, update vote counts, and visualize the progress of the voting	Flask, HTML/CSS
Online Bookstore	Product Listings: Display books available for sale, including book titles, authors, descriptions, and prices	Django, SQLite
	Shopping Cart: Allow users to add books to the cart, update quantities, and proceed with the checkout process	Django, HTML/CSS
	Order Management: Handle order placement, order confirmation, and generate invoices for successful transactions	Django, SQLite
Recipe Finder Application	Recipe Retrieval: Fetch recipes based on user input, including searching by ingredients, cuisine, or recipe name	Web scraping, APIs
	Recipe Details: Display recipe details such as ingredients, preparation steps, cooking time, and user ratings	Flask, HTML/CSS
	User Interface: Provide a user-friendly interface for searching, viewing, and saving recipes	Flask, HTML/CSS

Stock Market Analysis Tool	Data Retrieval: Retrieve stock market data from APIs and fetch information such as stock prices, historical data, and financial indicators	Financial APIs, Python requests
	Data Analysis: Perform analysis on stock data, including calculating moving averages, identifying trends, and generating statistical insights	Python libraries for data analysis
	Visualization: Present stock market trends and analysis using charts, graphs, and visual representations	Matplotlib, Plotly
Movie Recommendation System	Recommendation Engine: Suggest movies to users based on their preferences, viewing history, and ratings	Machine Learning algorithms, Python
	Movie Database: Store movie information, including titles, genres, ratings, and user reviews	Flask, HTML/CSS
	User Interface: Provide movie recommendations, display movie details, and allow users to rate and review movies	Flask, HTML/CSS
Expense Tracker	Expense Tracking: Allow users to add, categorize, and track personal or business expenses	Flask, SQLite
	Budget Management: Set budgets, track spending against budgets, and	Flask, HTML/CSS

	provide alerts for budget thresholds	
	Reporting: Generate expense reports, visualize spending patterns, and provide insights on expenditure trends	Flask, HTML/CSS
Blogging Platform	Blog Management: Create, publish, and manage blog posts, including features like drafts, editing, and comments	Django, SQLite
	User Authentication: Implement user registration, login, and profile management functionalities	Django, HTML/CSS
	User Interaction: Enable users to like, share, and comment on blog posts, follow other bloggers, and receive notifications	Django, HTML/CSS
Event Management System	Event Creation: Allow organizers to create and manage events, including setting event details, date, time, and venue	Django, SQLite
	Registration and Ticketing: Manage event registrations, generate tickets, and handle payment processing	Django, HTML/CSS
	Attendee Management: Track attendee information, send event updates, and generate attendee lists and reports	Django, SQLite

Online Auction System	Auction Management: Organize and manage online auctions, including creating auction listings, setting bid durations, and managing bids	Django, SQLite
	User Authentication: Implement user registration, login, and profile management functionalities	Django, HTML/CSS
	Bid Management: Enable users to place bids, track bid statuses, and notify users about outbid or winning bids	Django, HTML/CSS
Music Streaming Application	Music Playback: Stream and play music from a database or online sources, allowing users to create playlists and control playback	Flask, HTML/CSS, JavaScript
	User Interface: Provide a music player interface with features like play, pause, skip, and volume control	Flask, HTML/CSS
Online Job Portal	Job Listings: Display available job vacancies, including job titles, descriptions, requirements, and application deadlines	Django, SQLite
	Resume Submission: Allow job seekers to upload resumes, cover letters, and application materials	Django, HTML/CSS
	Employer Interaction: Provide features for	Django, HTML/CSS

	employers to post jobs, review applications, and communicate with applicants	

Note: The table includes an expanded list of projects, detailed module-wise use-cases, and the corresponding technologies/tools/APIs required for each project.

Software Project Development Process

To successfully complete the above projects within the given time frame and learn about software project development, students can follow a step-by-step process based on the Software Development Life Cycle (SDLC), incorporating various design concepts. Here's an extended version of the process:

1. Project Understanding and Planning:
    - Thoroughly understand the project requirements and objectives.
    - Identify the stakeholders and their needs.
    - Define project scope, constraints, and deliverables.
    - Create a project plan with specific milestones and deadlines.
2. Requirement Gathering and Analysis:
    - Conduct detailed requirement gathering sessions with stakeholders.
    - Document user stories, use cases, and functional requirements.
    - Use techniques like interviews, surveys, and brainstorming to gather requirements.
    - Analyze the gathered requirements and create a requirement specification document.
3. Design Phase:
    - Use the gathered requirements to design the system architecture.
    - Design the database schema using Entity-Relationship (ER) diagrams.
    - Create Data Flow Diagrams (DFDs) to illustrate data movement and processing within the system.
    - Design user interfaces using wireframes and mock-ups.
    - Apply user interface design principles to ensure usability and accessibility.
4. Development Iterations:
    - Break down the project into smaller tasks or user stories.
    - Prioritize and tackle the tasks based on their importance and dependencies.
    - Follow an iterative development approach, implementing one feature or module at a time.
    - Write clean, modular, and well-commented code using best practices.
5. Testing and Debugging:

- Conduct various types of testing, such as unit testing, integration testing, and system testing.
- Write test cases to verify the functionality and identify any defects.
- Perform regression testing to ensure that new changes do not impact existing features.
- Debug and fix any issues or bugs that arise during testing.

6. Integration and Deployment:
    - Integrate the different components or modules to ensure they work together seamlessly.
    - Deploy the application to a development or testing environment.
    - Perform system-level testing and user acceptance testing.
    - Prepare the application for production deployment.
7. Documentation:
    - Document the project, including its architecture, design, and usage instructions.
    - Create user manuals, technical documentation, and system documentation.
    - Document any APIs, libraries, or tools used in the project.
    - Include installation and configuration instructions for the application.
8. Presentation and Demonstration:
    - Prepare a presentation or demo to showcase the completed project.
    - Explain the project's features, functionality, and implementation details.
    - Use diagrams, such as ER diagrams and DFDs, to illustrate the system design and data flow.
    - Discuss the user interface design choices and how they enhance user experience.
9. Reflection and Learning:
    - Reflect on the project development process and identify areas for improvement.
    - Evaluate the challenges faced, lessons learned, and skills acquired.
    - Consider feedback from stakeholders and users for future enhancements.
    - Document your learnings and make note of any future improvements to the project.

By following this comprehensive process, incorporating the SDLC stages and design concepts like use case design, ER diagrams, data flow diagrams, and user interface design, students can effectively complete the projects within the given time frame while gaining valuable experience in software project development.

Contact