

## ASSIGNMENT – 11.1

NAME:CH.Charan Kumar

Roll.no:2303A54006

BATCH:47-A

### Lab 11 – Data Structures with AI: Implementing Fundamental Structures

#### **Lab Objectives**

- Use AI to assist in designing and implementing fundamental data structures in Python.
- Learn how to prompt AI for structure creation, optimization, and documentation.
- Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.
- Enhance code quality with AI-generated comments and performance suggestions.

#### **Task Description #1 – Stack Implementation**

Task: Use AI to generate a Stack class with push, pop, peek, and is\_empty methods.

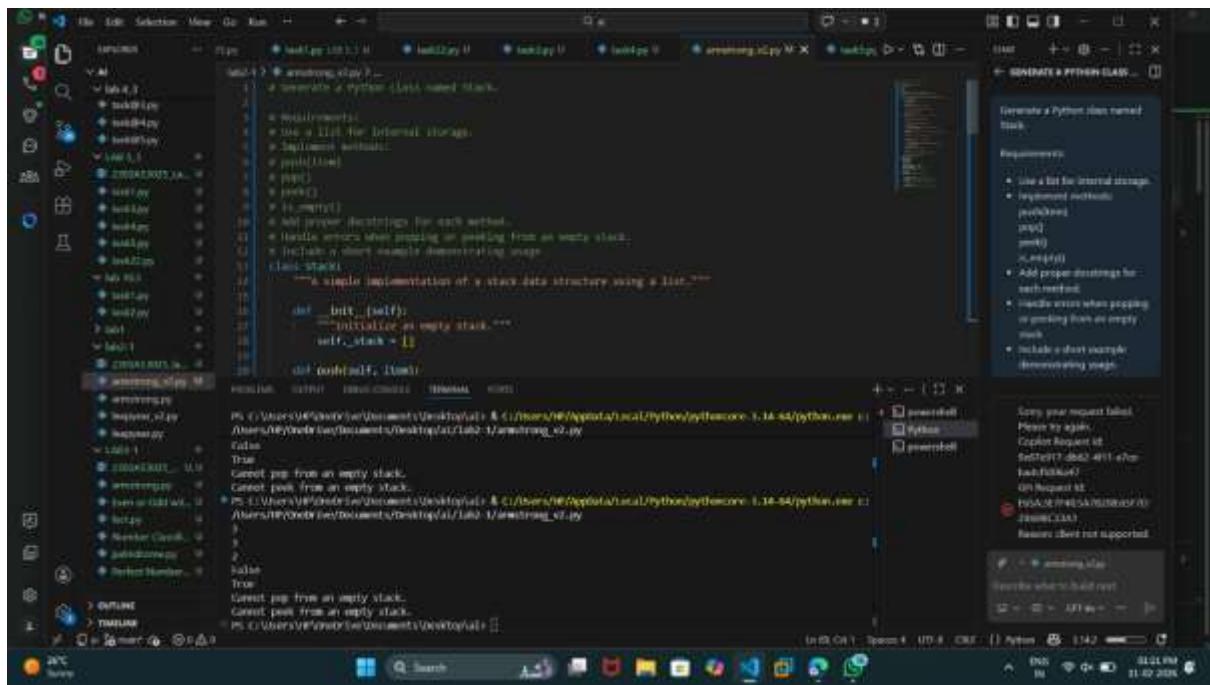
Sample Input Code:

```
class Stack:
```

```
    pass
```

Expected Output:

- A functional stack implementation with all required methods and docstrings.



## Task Description #2 – Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

```
class Queue:
```

```
    pass
```

Expected Output:

- FIFO-based queue class with enqueue, dequeue, peek, and size

Methods

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar with a tree view of files and folders. The main editor area contains Python code for a singly linked list. A status bar at the bottom indicates the file is 1142 bytes long and was last modified at 11:23 PM on 11-09-2023. A right-hand sidebar titled 'GENERATE A PYTHON CLASS...' displays a task description and requirements for generating a linked list class named 'Stack'.

```
print("Initial state: ", queue.dequeue())
print("Queue size after dequeue: ", queue.size())
print("Queue size after dequeue: ", queue.size())

try:
    empty_queue = Queue()
    empty_queue.dequeue()
except IndexError as e:
    print(e) # outputs: IndexError: pop from an empty queue

empty_queue.push(1)
# this will raise an error
except IndexError as e:
    print(e) # outputs: push from an empty queue
```

**Task Description #3 – Linked List**

**Task:** Use AI to generate a Singly Linked List with insert and display methods.

**Sample Input Code:**

class Node:

pass

class LinkedList:

pass

**Expected Output:**

- A working linked list implementation with clear method documentation.

The screenshot shows a code editor window with several tabs open. The main tab displays Python code for a singly linked list. The code includes a class definition for a node, which has a data attribute and a next attribute pointing to the next node. It also includes methods for inserting a new node at the head and printing the list. A sidebar on the right contains a 'GENERATE A PYTHON CLASS...' button and a 'Requirements' section with bullet points such as 'use a list for internal storage', 'implement methods: \_\_init\_\_(self, data), \_\_str\_\_(self)', and 'Add proper exception handling'.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def insert_at_head(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def __str__(self):
        current_node = self.head
        result = []
        while current_node:
            result.append(str(current_node.data))
            current_node = current_node.next
        return ' '.join(result)
```

## Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

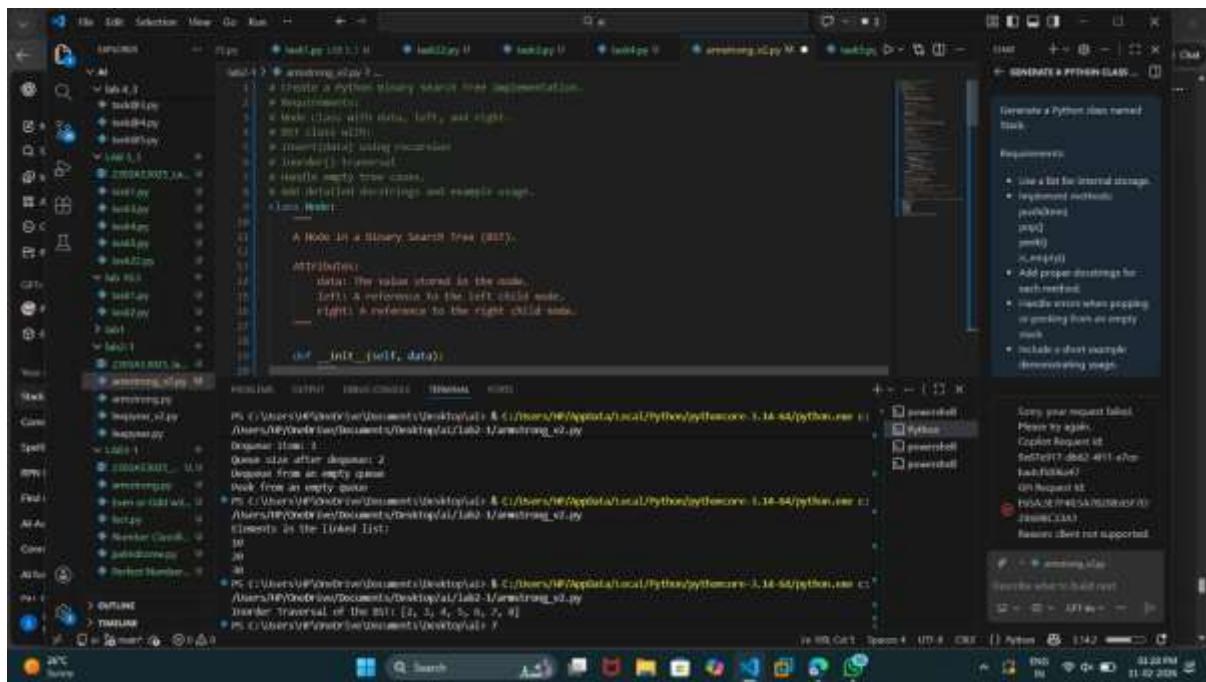
Sample Input Code:

```
class BST:
```

```
    pass
```

Expected Output:

- BST implementation with recursive insert and traversal method



## Task Description #5 – Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

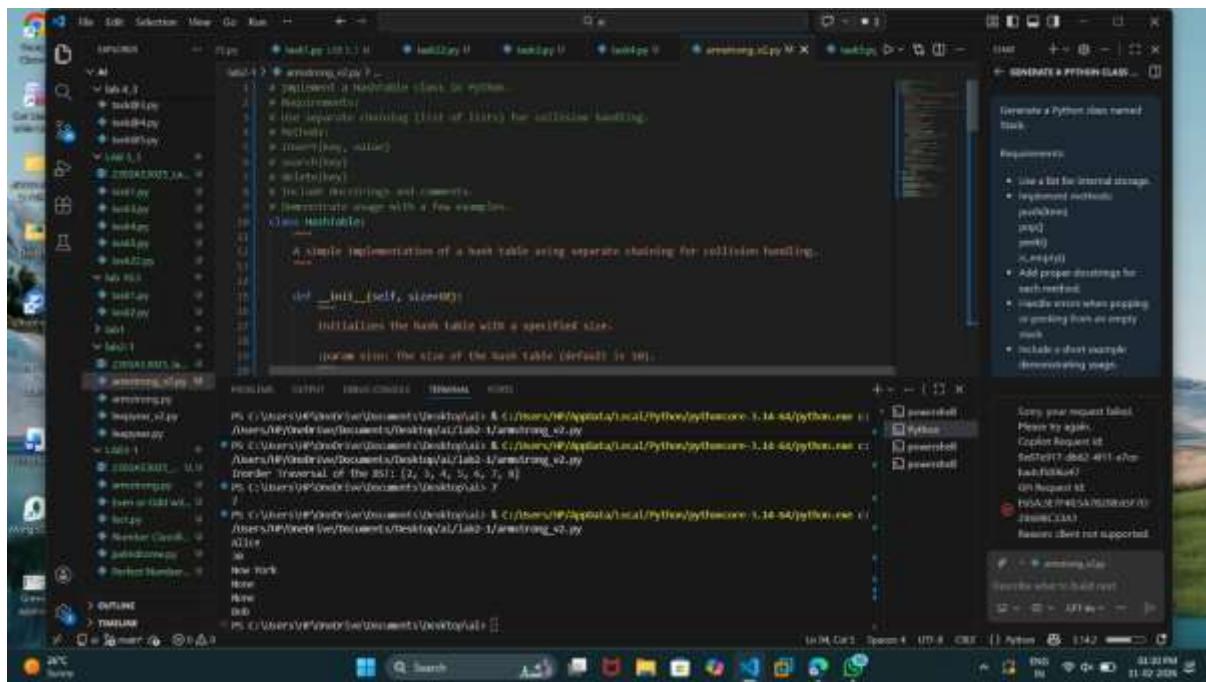
Sample Input Code:

class HashTable:

pass

Expected Output:

- Collision handling using chaining, with well-commented methods.



## Task Description #6 – Graph Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

class Graph:

pass

Expected Output:

- Graph with methods to add vertices, add edges, and display connections.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar with project files like `graph.py`, `testGraph.py`, and `testGraph_v2.py`. The main editor area displays `graph.py` with code for a graph class using an adjacency list representation. The terminal at the bottom shows a failed API request to `https://api.github.com/repos/microsoft/vscode/branches`.

```
graph.py
# A class to represent a graph using adjacency list representation.

class Graph:
    """A class to represent a graph using adjacency list representation."""

    def __init__(self):
        """Initialize the graph as an empty dictionary."""
        self.graph = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        self.graph[vertex] = []

# testGraph.py
import requests
import json

url = "https://api.github.com/repos/microsoft/vscode/branches"
response = requests.get(url)
print(response.status_code)
print(response.json())
print(response.headers)

# testGraph_v2.py
import requests
import json

url = "https://api.github.com/repos/microsoft/vscode/branches"
response = requests.get(url)
print(response.status_code)
print(response.json())
print(response.headers)

# terminal
PS C:\Users\MPV\OneDrive\Documents\GitHub\graph> python testGraph.py
404
[{"message": "Not Found"}, {"documentation_url": "https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404"}]
Content-Type: application/json; charset=utf-8
Date: Mon, 11 Jul 2022 14:47:41 GMT
Connection: keep-alive
Transfer-Encoding: chunked
```

## Task Description #7 – Priority Queue

Task: Use AI to implement a priority queue using Python's heapq module.

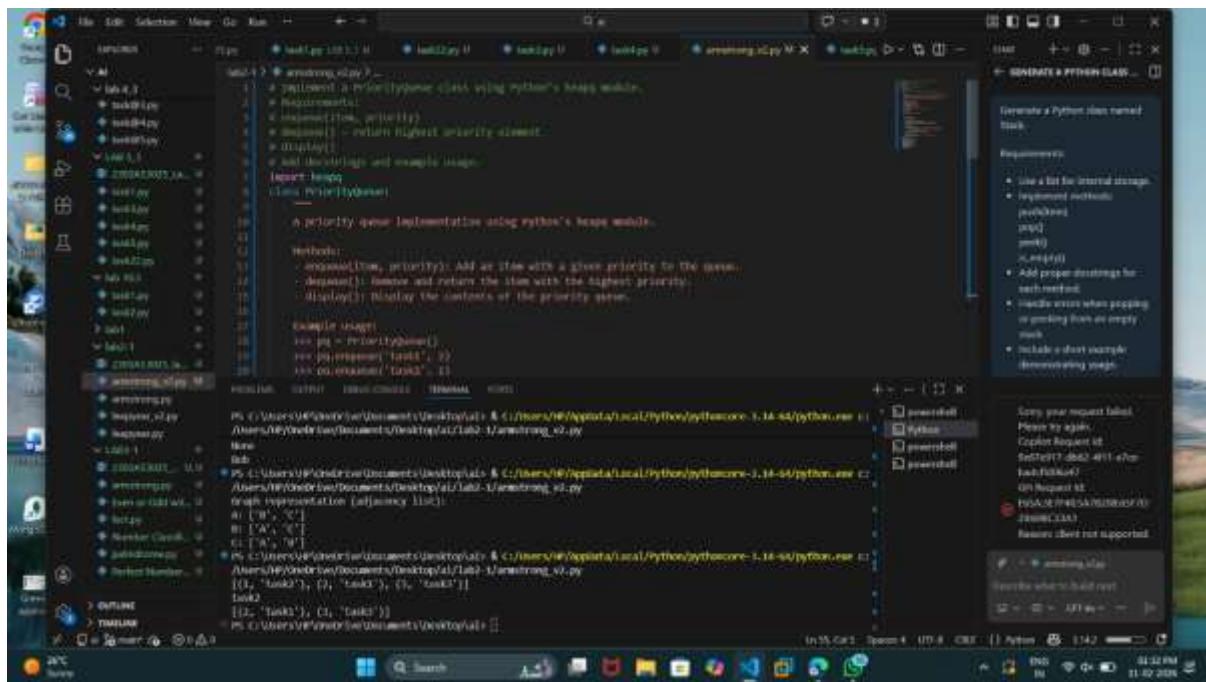
## Sample Input Code:

class PriorityQueue:

pass

## Expected Output:

- Implementation with enqueue (priority), dequeue (highest priority), and display methods.



## Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

```
class DequeDS:
```

```
    pass
```

Expected Output:

- Insert and remove from both ends with docstrings.

```
class Deque:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def add_front(self, item):
        self.items.append(item)

    def add_rear(self, item):
        self.items.insert(0, item)

    def delete_front(self):
        if self.is_empty():
            return None
        else:
            return self.items.pop()

    def delete_rear(self):
        if self.is_empty():
            return None
        else:
            return self.items.pop(0)
```

## Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure

Scenario:

Your college wants to develop a Campus Resource Management System that handles:

1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.
5. Cafeteria Order Queue – Serve students in the order they arrive.

Student Task:

- For each feature, select the most appropriate data structure from the list below:

- Stack
- Queue
- Priority Queue
- Linked List
- Binary Search Tree (BST)
- Graph
- Hash Table
- Deque

- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature with comments and docstrings.

The screenshot shows the PyCharm IDE interface with several tabs open at the top: 'Untitled 1', 'Untitled 2', 'Untitled 3', 'Untitled 4', 'Untitled 5', 'Untitled 6', 'Untitled 7', and 'Untitled 8'. The main editor window displays a Python script with code related to student attendance tracking. A code completion dropdown menu is open over the word 'Attendance' in the line 'class Attendance(ABC):'. The dropdown lists several suggestions:

- Attendance
- AttendanceTable
- AttendanceTableModel
- AttendanceTableWidget
- AttendanceTableWithActions
- AttendanceTableWithActionsModel
- AttendanceTableWithActionsWidget
- AttendanceTableWithActionsWithActions
- AttendanceTableWithActionsWithActionsModel
- AttendanceTableWithActionsWithActionsWidget

The PyCharm interface includes a 'File' menu at the top left, a 'Search' bar at the bottom center, and various toolbars and status bars along the bottom.

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The left sidebar shows a project named "Attendance". Inside, there's a package named "attendance" containing files like "attendance.py", "attendance.pyc", and "attendance.pycache".
- Code Editor:** The main window displays the content of "attendance.py".

```
class AttendanceTracker:  
    """A class to manage student attendance tracking using a queue.  
    """  
  
    def __init__(self):  
        """Initialize an empty queue for attendance tracking.  
        """  
        self.attendance_queue = []  
  
    def log_entry(self, student_name):  
        """Log a student's entry to the campus.  
  
        Args:  
            student_name (str): The name of the student entering the campus.  
        """  
        self.attendance_queue.append(student_name)  
        print(f"{student_name} has entered the campus.")  
  
    def log_exit(self):  
        """Log a student's exit from the campus.  
  
        Returns:  
            str: The name of the student exiting the campus, or None if no students are present.  
        """  
        if not self.attendance_queue:  
            print("No students are currently on campus.")  
            return None  
        exiting_student = self.attendance_queue.pop(0)  
        print(f"{exiting_student} has exited the campus.")  
        return exiting_student
```
- Run Tab:** The bottom tab bar shows the current file is "attendance.py". The status bar indicates the code has been successfully run.
- Right Panel:** A floating panel on the right provides options for generating Python classes and methods, such as "Generate Python class named 'Attendance'", "Requirements", "Implement methods", and "Add property".

## Task Description #10: Smart E-Commerce Platform – Data Structure

## Challenge

An e-commerce company wants to build a Smart Online Shopping System

with:

1. Shopping Cart Management – Add and remove products dynamically.
2. Order Processing System – Orders processed in the order they are placed.
3. Top-Selling Products Tracker – Products ranked by sales count.
4. Product Search Engine – Fast lookup of products using product ID.
5. Delivery Route Planning – Connect warehouses and delivery locations.

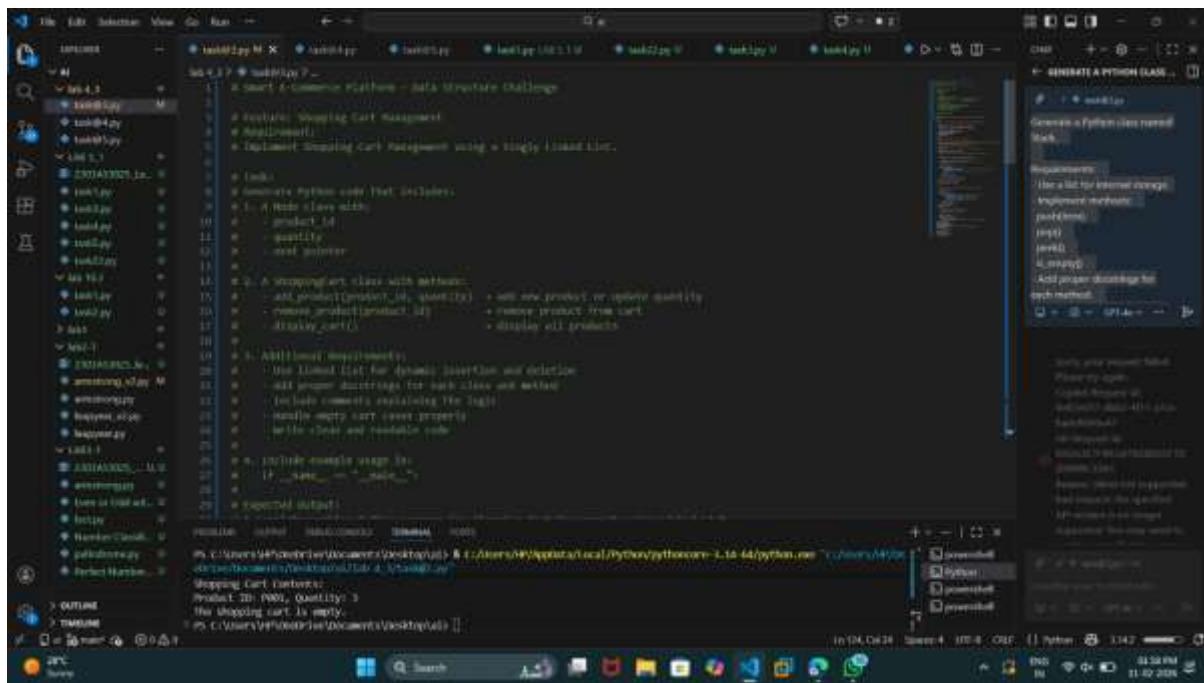
Student Task:

- For each feature, select the most appropriate data structure from the list below:
  - o Stack
  - o Queue
  - o Priority Queue
  - o Linked List
  - o Binary Search Tree (BST)
  - o Graph
  - o Hash Table
  - o Deque
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature

with comments and docstrings.



The screenshot shows the PyCharm IDE interface with the following details:

- File Structure:** On the left, the project tree shows files like `task01.py`, `task02.py`, `task03.py`, `task04.py`, `task05.py`, `task06.py`, `task07.py`, `task08.py`, `task09.py`, `task10.py`, `task11.py`, `task12.py`, `task13.py`, `task14.py`, `task15.py`, `task16.py`, `task17.py`, `task18.py`, `task19.py`, `task20.py`, `task21.py`, `task22.py`, `task23.py`, `task24.py`, `task25.py`, `task26.py`, `task27.py`, `task28.py`, `task29.py`, `task30.py`, `task31.py`, `task32.py`, `task33.py`, `task34.py`, `task35.py`, `task36.py`, `task37.py`, `task38.py`, `task39.py`, `task40.py`, `task41.py`, `task42.py`, `task43.py`, `task44.py`, `task45.py`, `task46.py`, `task47.py`, `task48.py`, `task49.py`, `task50.py`, `task51.py`, `task52.py`, `task53.py`, `task54.py`, `task55.py`, `task56.py`, `task57.py`, `task58.py`, `task59.py`, `task60.py`, `task61.py`, `task62.py`, `task63.py`, `task64.py`, `task65.py`, `task66.py`, `task67.py`, `task68.py`, `task69.py`, `task70.py`, `task71.py`, `task72.py`, `task73.py`, `task74.py`, `task75.py`, `task76.py`, `task77.py`, `task78.py`, `task79.py`, `task80.py`, `task81.py`, `task82.py`, `task83.py`, `task84.py`, `task85.py`, `task86.py`, `task87.py`, `task88.py`, `task89.py`, `task90.py`, `task91.py`, `task92.py`, `task93.py`, `task94.py`, `task95.py`, `task96.py`, `task97.py`, `task98.py`, `task99.py`, `task100.py`.
- Code Editor:** The main window displays Python code for a `ShoppingCart` class. The code includes methods for adding products to the cart, displaying the cart, and checking if it's empty.
- Code Completion:** A floating code completion window is open over the code editor, showing suggestions for methods like `add_product`, `remove_product`, `display_cart`, and `is_empty`. It also lists requirements such as `product`, `product_id`, `quantity`, and `unit`.
- Status Bar:** The bottom status bar shows system information including battery level (28%), CPU usage (100%), memory (8GB), and network (WIFI).