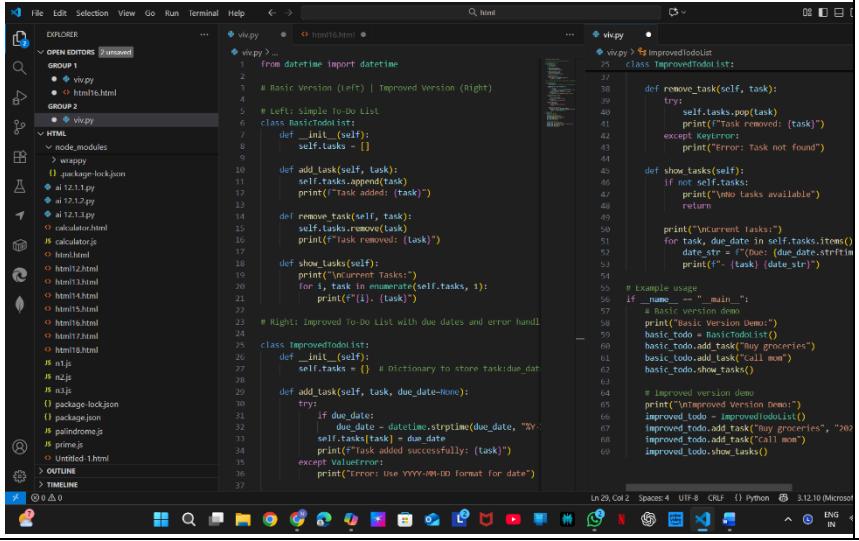
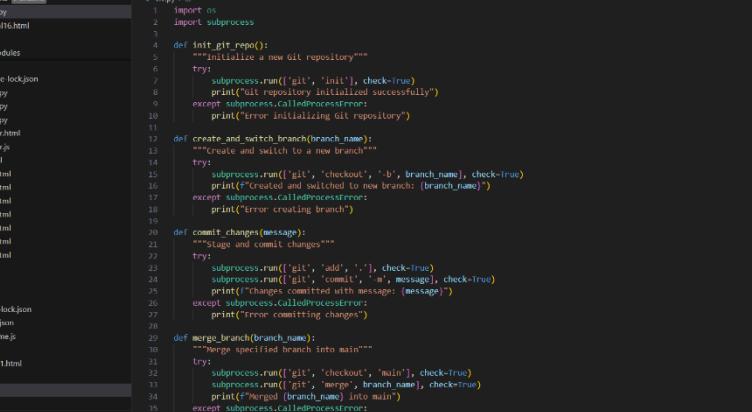


SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week11 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber: 21.1(Present assignment number)/ 24 (Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	Lab 21 – Collaborative AI Coding: Pair Programming and Version Control Integration Lab Objectives: <ul style="list-style-type: none"> • Learn to use AI as a pair programmer for writing, reviewing, and optimizing code. • Understand how to integrate AI-generated code into version 		Week11 - Monday

	<p>control systems (Git).</p> <ul style="list-style-type: none"> • Practice collaborative workflows such as commit messages, branching, and merging. 	
	<p>Task 1 – AI-Assisted Pair Programming</p> <ul style="list-style-type: none"> • Use AI to generate an initial program (e.g., a simple To-Do List Manager). • Ask AI to suggest enhancements (such as error handling, additional features, or refactoring). • Compare the original and improved versions to understand the role of AI in collaboration.  <pre> File Edit Selection View Go Run Terminal Help <- > O html EXPLORER OPEN EDITORS 2 unsaved GROUP 1 vivpy vivpy.py html1.html GROUP 2 vivpy vivpy.py HTML vivpy.html node_modules vivpy package-lock.json ai 12.1.py ai 12.2.py ai 12.3.py calculator.html calculator.js html1.html html2.html html3.html html4.html html5.html html6.html html7.html html8.html html9.html n1.js n2.js n3.js package-lock.json package.json palindrome.js prime.js Untitled 1.html OUTLINE TIMELINE ● 0:0.0 vivpy vivpy vivpy ImprovedTodoList 1 from datetime import datetime 2 3 # Basic Version (left) Improved Version (right) 4 5 # Left: Simple To-Do List 6 class BasicTodoList: 7 def __init__(self): 8 self.tasks = [] 9 10 def add_task(self, task): 11 self.tasks.append(task) 12 print(f"Task added: {task}") 13 14 def remove_task(self, task): 15 self.tasks.remove(task) 16 print(f"Task removed: {task}") 17 18 def show_tasks(self): 19 print("Current Tasks:") 20 for i, task in enumerate(self.tasks, 1): 21 print(f"{i}. {task}") 22 23 # Right: Improved To-Do List with due dates and error handling 24 25 class ImprovedTodoList: 26 def __init__(self): 27 self.tasks = {} # Dictionary to store tasks:due_date 28 29 def add_task(self, task, due_date=None): 30 try: 31 if due_date: 32 due_date = datetime.strptime(due_date, "%Y-%m-%d") 33 self.tasks[task] = due_date 34 print(f"Task added successfully: {task}") 35 except ValueError: 36 print("Error: Use YYYY-MM-DD format for date") 37 38 def remove_task(self, task): 39 try: 40 self.tasks.pop(task) 41 print(f"Task removed: {task}") 42 except KeyError: 43 print("Error: Task not found") 44 45 def show_tasks(self): 46 if not self.tasks: 47 print("No tasks available") 48 return 49 50 print("Current Tasks:") 51 for task, due_date in self.tasks.items(): 52 date_str = f"(Due: {due_date.strftime('%Y-%m-%d')})" 53 print(f"- {task} {date_str}") 54 55 # Example usage 56 if __name__ == "__main__": 57 # Basic version demo 58 print("Basic Version Demo:") 59 basic_todo = BasicTodoList() 60 basic_todo.add_task("Buy groceries") 61 basic_todo.add_task("Call mom") 62 basic_todo.show_tasks() 63 64 # Improved version demo 65 print("Improved Version Demo:") 66 improved_todo = ImprovedTodoList() 67 improved_todo.add_task("Buy groceries", "2023-12-25") 68 improved_todo.add_task("Call mom") 69 improved_todo.show_tasks() </pre>	

	<pre> PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PS C:\Users\vivek\html> & C:/Users/vivek/AppData/Local/Microsoft/Windows/PowerShell/Scripts/todo.py Basic Version Demo: Task added: Buy groceries Task added: Call mom Current Tasks: 1. Buy groceries 2. Call mom Improved Version Demo: Task added successfully: Buy groceries Task added successfully: Call mom Current Tasks: - Buy groceries (Due: 2024-01-15) Current Tasks: Current Tasks: - Buy groceries (Due: 2024-01-15) - Call mom (No due date) PS C:\Users\vivek\html> </pre> <ul style="list-style-type: none"> • • Explanation: Defines two to-do implementations: BasicTodoList (uses a list) and ImprovedTodoList (uses a dict mapping task -> due_date). • BasicTodoList: add_task/appends, remove_task/removes (no error handling), show_tasks prints numbered list. • ImprovedTodoList: add_task accepts optional due_date string, parses YYYY-MM-DD to datetime and stores it; prints error on invalid date. • Improved remove_task catches missing tasks and prints an error; show_tasks lists tasks with formatted due dates or “No due date”. • If run as main, the script demos both versions by adding tasks and printing their contents. • 	
	<p>Task 2 – Version Control Integration</p> <ul style="list-style-type: none"> • Initialize a Git repository for your project. • Perform the following steps: <ol style="list-style-type: none"> 1. Create and switch to a new branch. 2. Add AI-assisted modifications. 3. Commit with a meaningful message. 4. Merge changes back into the main branch. 	

- Reflect on how version control supports collaborative development with AI assistance.



A screenshot of a Windows desktop environment. In the center, a terminal window titled 'viv.py' is open, displaying Python code for managing a Git repository. The code includes functions for initializing a repository, creating branches, committing changes, and merging branches. On the left, a file explorer sidebar shows a project structure with files like 'viv.py', 'viv.html', and various JSON configuration files. At the bottom, the taskbar displays icons for various applications including Microsoft Edge, File Explorer, and the Start button.

```
viv.py
1 import os
2 import subprocess
3
4 def init_git_repo():
5     """Initialize a new Git repository"""
6     try:
7         subprocess.run(['git', 'init'], check=True)
8         print("Git repository initialized successfully")
9     except subprocess.CalledProcessError:
10        print("Error initializing Git repository")
11
12 def create_and_switch_branch(branch_name):
13     """Create and switch to a new branch"""
14     try:
15         subprocess.run(['git', 'checkout', '-b', branch_name], check=True)
16         print(f"Created and switched to new branch: {branch_name}")
17     except subprocess.CalledProcessError:
18        print("Error creating branch")
19
20 def commit_changes(message):
21     """Stage and commit changes"""
22     try:
23         subprocess.run(['git', 'add', '.'], check=True)
24         subprocess.run(['git', 'commit', '-m', message], check=True)
25         print(f"Changes committed with message: {message}")
26     except subprocess.CalledProcessError:
27        print("Error committing changes")
28
29 def merge_branch(branch_name):
30     """Merge specified branch into main"""
31     try:
32         subprocess.run(['git', 'checkout', 'main'], check=True)
33         subprocess.run(['git', 'merge', branch_name], check=True)
34         print(f"Merged {branch_name} into main")
35     except subprocess.CalledProcessError:
36        print("Error merging branch")
```

```
if __name__ == "__main__":
    # Initialize repository
    init_git_repo()

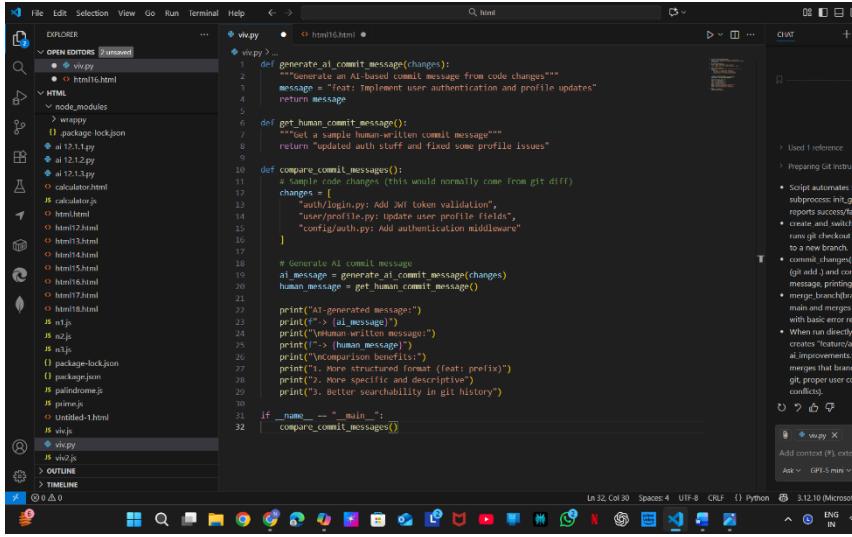
    # Create and switch to feature branch
    create_and_switch_branch("feature/ai-improvements")

    # Make some changes and commit
    with open("ai_improvements.txt", "w") as f:
        f.write("AI-assisted code improvements")

    commit_changes("Add AI-assisted improvements")

    # Merge changes back to main
    merge_branch(["feature/ai-improvements"])
```

- Explanation: Script automates simple Git tasks via subprocess:
init_git_repo() runs git init and reports success/failure.
 - create_and_switch_branch(branch_name) runs git checkout -b to create and switch to a new branch.
 - commit_changes(message) stages all files (git add .) and commits with the provided message, printing an error on failure.
 - merge_branch(branch_name) checks out main and merges the given branch into it, with basic error reporting.
 - When run directly it initializes a repo, creates "feature/ai-improvements", writes ai_improvements.txt, commits it, then merges that branch into main (requires git, proper user config, and may fail on conflicts).

	<p>Task 3 – AI for Commit Message Generation</p> <ul style="list-style-type: none"> • Use AI to generate concise, meaningful commit messages based on the changes made. • Compare AI-generated messages with human-written ones. • Discuss how AI can improve documentation and traceability in Git history.  <ul style="list-style-type: none"> • Explanation: <code>generate_ai_commit_message(changes)</code> is a stub that returns a fixed, structured commit message (would normally derive text from the changes). • <code>get_human_commit_message()</code> returns a sample human-written commit message. • <code>compare_commit_messages()</code> prepares a mock list of changed files, calls both generators, and prints the AI/human messages plus three brief claimed benefits. • The if <code>name == "main"</code> block runs <code>compare_commit_messages()</code> when the script is executed directly. • Note: AI generation is mocked — to be useful replace the stub with real diff parsing or an ML/model call for context-aware messages. • • 	
	<p>Task 4 – Pair Programming Simulation</p> <ul style="list-style-type: none"> • Student A writes the initial version of a program. • Student B, with AI assistance, suggests improvements or adds features. • Both students collaborate using Git branching and merging to 	

integrate their work.

```

File Edit Selection View Go Run Terminal Help ↵ → viv.py • html&html
OPEN EDITORS 2 unsaved
viv.py
html&html
HTML
node_modules
ai
ai package-lock.json
ai 12.1.js
ai 12.1.2.js
ai 12.1.3.js
calculator.html
calculator.js
html&html
html12.html
html13.html
html14.html
html15.html
html16.html
html17.html
html18.html
n1.js
n2.js
n3.js
package-lock.json
package.json
palindrome.js
prime.js
Untitled-1.html
viv.js

File Edit Selection View Go Run Terminal Help ↵ → viv.py • html&html
OPEN EDITORS 2 unsaved
viv.py
html&html
HTML
node_modules
ai
ai package-lock.json
ai 12.1.js
ai 12.1.2.js
ai 12.1.3.js
calculator.html
calculator.js
html&html
html12.html
html13.html
html14.html
html15.html
html16.html
html17.html
html18.html
n1.js
n2.js
n3.js
package-lock.json
package.json
palindrome.js
prime.js
Untitled-1.html
viv.js

In B1, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.10 (Microsoft)
In B2, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.10 (Microsoft)

```

- Explanation: Tiny CLI todo app (c:\Users\vivek\html\viv.py) that stores tasks in a JSON file next to the script (same name with .todos.json).
- load()/save() handle JSON persistence; add() creates a task with an 8-char UUID, priority, created_at (UTC ISO), done flag, and optional due.
- list_() prints tasks sorted by done, priority, and creation time; supports hiding completed tasks with --pending.
- update(id, **kw) edits fields if provided, remove(id) deletes a task, and stats() prints totals/done/pending.
- CLI wired with argparse: commands are add, list, done, remove, edit, and stats; main() dispatches and prints simple status messages.

	<p><input checked="" type="checkbox"/> Deliverables (For All Tasks)</p> <ol style="list-style-type: none">1. AI-generated prompts for code and test case generation.2. At least 3 assert test cases for each task.3. AI-generated initial code and execution screenshots.4. Analysis of whether code passes all tests.5. Improved final version with inline comments and explanation.6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.	
--	---	--