LADOKE AKINTOLA UNIVERSITY OF TECHNOLOGY

Computer Science

Lecturer: Mr Arowoyele Zacchaeus

Time Tracking and Productivity Analyzer

Course: CSC 202 - Computer Programming Language II

Group 30 Project: Time Tracking and Productivity Analyzer Build a productivity analysis tool that reads time tracking data from CSV files, calculates work hours and productivity metrics for different activities, analyzes time distribution patterns across projects and tasks, implements break time optimization and efficiency calculations, track's goal achievement and deadline performance, and generates productivity reports with time management recommendations and efficiency improvement strategies

Group Members:

- Chukwunenye Victor Chinedu 2024010965
- Ojetola Seyi Emmanuel 2023005705
- KAYODE KOSISOCHUKWU HENRY 2023006084
- Ogundare faith Precious 2023002769
- Aremu Glory Opemipo 2023004359

Project Description

This CLI-based Python project is designed to help users track the amount of time spent on various tasks, monitor deadline performance, and generate basic productivity reports from CSV data. It emphasizes simplicity, efficiency, and accuracy in managing personal or group project timelines.

Key Features

CSV File Input: Accepts structured data on task names, start/end times, and deadlines.

Time Analysis: Computes total hours worked per project.

Deadline Evaluation: Identifies which tasks were completed on time or overdue.

Summary Report: Outputs the most time-consuming task and overall performance stats.

Command-Line Interface: Allows users to run functions via terminal with custom arguments.

Testing Support: Provides a set of automated unit tests to ensure code reliability.

User Manual

 # Time Tracking and Productivity Analyzer

This is a command-line Python tool to help you track how you spend your time, how productive you are, and how well you're meeting deadlines.

Features

- Load time tracking data from a CSV file
- Calculate total hours and productive hours
- Analyze time spent on each project
- Show how much break time you take
- Check how well you're meeting deadlines
- Give tips to improve productivity

Files

- `Time_Tracker.py` – Main program file
- `Tracker_Test.py` – Unit tests for main features
- `data1.csv` to `data5.csv` – Example CSV files with tracking data
- `User_Manual.md` – This user guide

CSV File Format

Each row should follow this format:

Start,End,Project,Deadline,Productive

2025-07-21 09:00,2025-07-21 11:00,AI Assistant,2025-08-01,Yes

- `Start` and `End` – The time you started and ended a task (use `YYYY-MM DD HH:MM`)
- `Project` – The project or task name
- `Deadline` – When the project is due (can be empty)
- `Productive` – Yes if it was productive time, No otherwise

How to Use

1. Open a terminal and navigate to the folder with `Time_Tracker.py`
2. To load data and view the productivity report:

   python Time_Tracker.py load --file data1.csv

Running Unit Tests

Run the test file to make sure everything works:

python Tracker_Test.py

Requirements

- Python 3.8 or later
- Uses only standard libraries: `csv`, `datetime`, `argparse`, `unittest`, `collections`

MAIN PROGRAM (Time_Tracker.py)

"""

Time Tracking and Productivity Analyzer

This CLI-based tool helps users track time spent on projects, calculate productivity, analyze deadline performance, and generate actionable reports.

Usage:

    python tracker.py load --file data.csv

    python tracker.py report

Department: Computer science

Group: 30

"""

```
# Import required libraries
from datetime import datetime,timedelta            # For working with date and time
import csv                                 # For reading CSV files
import argparse                            # For bcvuilding a command-line interface
from collections import defaultdict        # For default dictionary to accumulate durations

# Define the main class for the productivity tracker
```

```
class ProductivityTracker:
    def __init__(self):
        self.entries = []               # Stores all the task entries from the CSV
        self.Total_hours = 0            # Total time tracked
        self.Totalprod_hours = 0        # Total productive time
        self.efficiency = 0             # Efficiency percentage
        self.break_time = 0             # Total break time
        self.goals_percent = 0          # Percentage of productive goals achieved
        self.on_time_percent = 0        # Percentage of tasks completed on time
        self.most_time_project = ""     # Project with most time spent
        self.project_time = {}          # Dictionary of xg5   names and their durations
        self.project_deadlines = {}     # Dictionary of deadlines left for each project
        self.time_break = "medium"      # Break time rating: low, medium, or high

    # Load the CSV file into memory
    def load_csv(self, filepath):
        with open(filepath, 'r') as file:
            reader = csv.DictReader(file)     # Read CSV into dictionary rows
            for row in reader:
                try:
                    # Parse Start and End times into datetime objects
                    row['Start'] = datetime.strptime(row['Start'], "%Y-%m-%d %H:%M")
                    row['End'] = datetime.strptime(row['End'], "%Y-%m-%d %H:%M")
```

```
                # Parse deadline if it exists
                deadline = row.get('Deadline')
                if deadline:
                    row['Deadline'] = datetime.strptime(deadline, "%Y-%m-%d %H:%M")
                else:
                    row['Deadline'] = None

                # Calculate duration in hours
                row['Duration'] = (row['End'] - row['Start']).total_seconds() / 3600

                # Add the row to the entries list
                self.entries.append(row)
            except Exception as e:
                print(f'Skipping row due to error: {e}')  # Handle any errors in row
parsing

    # Calculate total and productive hours and efficiency
    def calculate_workhours(self):
        for entry in self.entries:
            self.Total_hours += entry['Duration']           # Sum all duration
            if entry['Productive'].strip().lower() == 'yes':
                self.Totalprod_hours += entry['Duration']     # Sum productive time
```

```
        if self.Total_hours > 0:
            self.efficiency = (self.Totalprod_hours / self.Total_hours) * 100  # Compute efficiency



    # Analyze how time is distributed across projects and determine break usage
    def project_analysis(self):
        project_time = defaultdict(float)      # Track total time per project
        project_prodtime = defaultdict(float)  # Track productive time per project

        for entry in self.entries:
            project = entry['Project']
            project_time[project] += entry['Duration']       # Add to total time
            if entry['Productive'].strip().lower() == 'yes':
                project_prodtime[project] += entry['Duration']  # Add to productive time

        self.project_time = dict(sorted(project_time.items()))  # Store sorted total time per project

        # Calculate productivity % per project (not used here, but could be printed)
        project_prodpercent = {
            project: (project_prodtime[project] / project_time[project]) * 100
            for project in project_time
        }
```

```
        # Handle break time logic
        for key in self.project_time:
            if key.lower() == 'break':
                self.break_time = self.project_time[key]

        # Calculate break ratio and categorize it
        break_ratio = (self.break_time / self.Total_hours) * 100 if self.Total_hours else
0
        if break_ratio > 25:
            self.time_break = 'high'
        elif break_ratio >= 20:
            self.time_break = 'medium'
        else:
            self.time_break = 'low'

        # Determine the project with the highest time spent
        self.most_time_project = max(self.project_time, key=self.project_time.get)

    # Evaluate goal achievement and deadline performance
    def goal_achievement(self):
        goals = 0        # Productive non-break entries
        on_time = 0      # Entries completed before deadline
```

```
        total = len(self.entries)

        for entry in self.entries:
            if entry['Project'].strip().lower() != 'break' and
entry['Productive'].strip().lower() == 'yes':
                goals += 1
            if entry['Deadline'] is not None and entry['End'] <= entry['Deadline']:
                on_time += 1

        if total > 0:
            self.goals_percent = (goals / total) * 100        # % of goals achieved
            self.on_time_percent = (on_time / total) * 100    # % of deadlines met

    # Calculate time remaining for each project deadline
    def project_deadline(self):
        self.project_deadlines = {}
        for entry in self.entries:
            if entry['Deadline'] is not None:
                diff = (entry['Deadline'] - entry['End']).total_seconds()
                self.project_deadlines[entry['Project']] = str(timedelta(seconds = diff))

    # Print the full productivity report
    def generate_report(self):
```

```
        print("\n PRODUCTIVITY REPORT")
        print(f"Total Time Tracked: {self.Total_hours:.2f} hrs")
        print(f"Productive Time: {self.Totalprod_hours:.2f} hrs")
        print(f"Non-Productive Time: {self.Total_hours - self.Totalprod_hours:.2f} hrs")
        print(f"Efficiency: {self.efficiency:.2f}%")
        print(f"Total Break Time: {self.break_time:.2f} hrs")
        print(f"Most Time Spent On: {self.most_time_project}
({self.project_time[self.most_time_project]:.2f} hrs)")

        print("\n TIME MANAGEMENT RECOMMENDATIONS")
        print(f"Break time: {self.break_time:.2f} hrs")
        if self.time_break == 'high':
            print("- Reduce break frequency to avoid time waste.")
        elif self.time_break == 'low':
            print("- Increase break frequency to avoid burnout.")
        if self.efficiency < 60:
            print("- Improve focus by batching similar tasks or reducing distractions.")

        print("\n EFFICIENCY IMPROVEMENT STRATEGIES")
        if self.goals_percent < 60:
            print("- Try setting more achievable goals.")
        if self.on_time_percent < 60:
            print("- Work more efficiently to meet deadlines.")
```

```
        print("- Use Pomodoro (25 min work + 5 min break).")
        print("- Review your top time-consuming projects weekly.")
        print("- Allocate productive hours to priority work.")
        print("- Avoid frequent task switching.")

        print("\n TIME REMAINING ON PROJECTS")
        for key, hours_left in self.project_deadlines.items():
            print(f"{key} -- {hours_left} hrs")

# CLI (Command Line Interface) entry point
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Time Tracking and Productivity Analyzer')  # Create CLI parser
    subparsers = parser.add_subparsers(dest='command')  # Add subcommands (load, report)

    # Subcommand: load CSV
    load_parser = subparsers.add_parser('load')        # Add 'load' command
    load_parser.add_argument('--file', required=True)   # Add --file argument

    # Subcommand: generate report
    subparsers.add_parser('report')                # Add 'report' command

    args = parser.parse_args()                     # Parse command-line arguments
```

```
tracker = ProductivityTracker()                 # Create an instance of the tracker

if args.command == 'load':                      # If user runs 'load' command
    tracker.load_csv(args.file)                 # Load the specified CSV file
    print(f"CSV loaded from {args.file}")       # Confirm load
    tracker.calculate_workhours()
    tracker.project_analysis()
    tracker.goal_achievement()
    tracker.project_deadline()
    tracker.generate_report()
```

UNIT TESTS(Tracker_Test.py)

```
# Import the unittest module for testing
import unittest

# Import datetime to create mock timestamps
from datetime import datetime

# Import the ProductivityTracker class from your main script
from Time_Tracker import ProductivityTracker

# Define a class to group all test cases related to ProductivityTracker
class TestProductivityTracker(unittest.TestCase):

    # Test whether work hours and efficiency are calculated correctly with valid data
    def test_load_valid_data(self):
        tracker = ProductivityTracker()  # Create an instance of the tracker

        # Manually create one time entry
        tracker.entries = [
            {
                'Start': datetime(2025, 7, 21, 9, 0),     # Start time: 9:00 AM
                'End': datetime(2025, 7, 21, 11, 0),      # End time: 11:00 AM
                'Project': 'AI Assistant',              # Project name
```

```
            'Deadline': datetime(2025, 8, 1),       # Deadline date
            'Productive': 'Yes',                    # Marked as productive
            'Duration': 2.0                         # Duration in hours
        }
    ]

    tracker.calculate_workhours()  # Calculate total and productive hours

    self.assertEqual(tracker.Total_hours, 2.0)        # Should match duration
    self.assertEqual(tracker.Totalprod_hours, 2.0)    # All hours are productive
    self.assertEqual(tracker.efficiency, 100.0)       # 100% efficiency

# Test how break time is handled and categorized
def test_break_time_analysis(self):
    tracker = ProductivityTracker()  # Create an instance of the tracker

    # Create one break entry and one productive entry
    tracker.entries = [
        {
            'Start': datetime(2025, 7, 21, 13, 0),    # 1:00 PM
            'End': datetime(2025, 7, 21, 13, 30),     # 1:30 PM
            'Project': 'Break',                       # Marked as break
            'Deadline': None,                         # No deadline
```

```
            'Productive': 'No',                  # Not productive
            'Duration': 0.5                      # 30 minutes break
        },
        {
            'Start': datetime(2025, 7, 21, 14, 0),    # 2:00 PM
            'End': datetime(2025, 7, 21, 16, 0),      # 4:00 PM
            'Project': 'AI Assistant',
            'Deadline': datetime(2025, 8, 1),
            'Productive': 'Yes',
            'Duration': 2.0
        }
    ]

    tracker.calculate_workhours()  # Sum total and productive hours
    tracker.project_analysis()     # Analyze break and work distribution

    self.assertEqual(tracker.break_time, 0.5)         # Break time is 0.5 hrs
    self.assertEqual(tracker.time_break, 'medium')       # 0.5 / 2.5 = 20% → low

# Test goal achievement and on-time completion percentages
def test_goal_achievement(self):
    tracker = ProductivityTracker()  # Create an instance of the tracker

    # Two entries: one productive work and one break
    tracker.entries = [
```

```
            {
                'Start': datetime(2025, 7, 21, 10, 0),    # 10:00 AM
                'End': datetime(2025, 7, 21, 12, 0),      # 12:00 PM
                'Project': 'Game Engine',
                'Deadline': datetime(2025, 7, 22),
                'Productive': 'Yes',
                'Duration': 2.0
            },
            {
                'Start': datetime(2025, 7, 21, 12, 30),
                'End': datetime(2025, 7, 21, 13, 0),
                'Project': 'Break',
                'Deadline': None,
                'Productive': 'No',
                'Duration': 0.5
            }
        ]

        tracker.goal_achievement()  # Analyze goals met and deadline performance

        self.assertEqual(tracker.goals_percent, 50.0)      # 1 out of 2 productive goals
        self.assertEqual(tracker.on_time_percent, 50.0)    # 1 entry met deadline

# Run the unit tests when this script is executed directly
if __name__ == '__main__':
    unittest.main()
```

CSV FILES

Start,End,Project,Deadline,Productive

2025-07-21 09:00,2025-07-21 11:00,AI Assistant,2025-08-01 12:00,Yes

2025-07-21 11:15,2025-07-21 12:00,Game Engine,2025-08-10 12:00,Yes

2025-07-21 13:00,2025-07-21 13:30,Break,,No

2025-07-21 14:00,2025-07-21 15:30,Portfolio Website,2025-08-05 12:00,Yes 2025-07-21 16:00,2025-07-21 17:00,Social Media Research,2025-07-30 12:00,No

2025-07-22 09:00,2025-07-22 10:30,AI Assistant,2025-08-01 12:00,Yes

2025-07-22 11:00,2025-07-22 12:30,Game Engine,2025-08-10 12:00,Yes

2025-07-22 13:30,2025-07-22 14:00,Break,,No

Start,End,Project,Deadline,Productive

2025-07-25 09:00,2025-07-25 11:00,Game Engine,2025-08-10 12:00,Yes

2025-07-25 11:15,2025-07-25 12:45,AI Assistant,2025-08-01 12:00,Yes

2025-07-25 13:15,2025-07-25 13:45,Break,,No

2025-07-25 14:00,2025-07-25 15:00,Portfolio Website,2025-08-05 12:00,Yes

2025-07-25 15:30,2025-07-25 16:30,Social Media Research,2025-07-30 12:00,No

Start,End,Project,Deadline,Productive

2025-07-24 08:30,2025-07-24 10:00,AI Assistant,2025-08-01 12:00,Yes

2025-07-24 10:15,2025-07-24 11:30,Portfolio Website,2025-08-05 12:00,Yes

2025-07-24 12:00,2025-07-24 12:30,Break,,No

2025-07-24 13:00,2025-07-24 14:45,Game Engine,2025-08-10 12:00,Yes

2025-07-24 15:00,2025-07-24 16:30,Social Media Research,2025-07-30 12:00,No

Start,End,Project,Deadline,Productive

2025-07-25 09:00,2025-07-25 11:00,Game Engine,2025-08-10 12:00,Yes

2025-07-25 11:15,2025-07-25 12:45,AI Assistant,2025-08-01 12:00,Yes

2025-07-25 13:15,2025-07-25 13:45,Break,,No

2025-07-25 14:00,2025-07-25 15:00,Portfolio Website,2025-08-05 12:00,Yes

2025-07-25 15:30,2025-07-25 16:30,Social Media Research,2025-07-30 12:00,No

Start,End,Project,Deadline,Productive

2025-07-26 09:30,2025-07-26 10:45,AI Assistant,2025-08-01 12:00,Yes

2025-07-26 11:00,2025-07-26 12:30,Game Engine,2025-08-10 12:00,Yes

2025-07-26 13:00,2025-07-26 13:30,Break,,No

2025-07-26 14:00,2025-07-26 15:15,Portfolio Website,2025-08-05 12:00,Yes

2025-07-26 15:30,2025-07-26 16:30,Social Media Research,2025-07-30 12:00,No

SAMPLE OUTPUT

python Time tracker.py load-file datal.csv

CSV loaded from datal.csv

PRODUCTIVITY REPORT

Total Time Tracked: 9.25 hrs

Productive Time: 7.25 hrs

Non-Productive Time: 2.00 hrs

Efficiency: 78.38%

Total Break Time: 1.00 hrs

Most Time Spent On: AI Assistant (3.50 hrs).

TIME MANAGEMENT RECOMMENDATIONS

Break time: 1.00 hrs

-Increase break frequency to avoid burnout.

EFFICIENCY IMPROVEMENT STRATEGIES

-Use Pomodoro (25 min work +5 min break).

-Review your top time-consuming projects weekly.

-Allocate productive hours to priority work.

-Avoid frequent task switching.

TIME REMAINING ON PROJECTS

Al Assistant-10 days, 1:30:00 hrs

Game Engine-18 days, 23:30:00 hrs

Portfolio Website-14 days, 20:30:00 hrs

Social Media Research-8 days, 19:00:00 hrs

NOTES AND REFERENCES

Modules and Libraries Used

datetime

Used for handling and formatting timestamps. Essential for calculating time differences (e.g., durations, deadlines, and productivity).

timedelta (from datetime)

Enables arithmetic operations between date and time objects, such as calculating the difference between start and end times.

csv

Handles reading from and writing to CSV files. Used to load project time-tracking data from external files.

argparse

Builds a command-line interface (CLI) for the tool. Allows the user to run commands like:

python tracker.py load --file data.csv

collections.defaultdict

Simplifies grouping and counting data without needing to check if keys exist beforehand.

unittest

Python's built-in testing framework. Used to write and run automated tests that validate the correctness of the project's functionality.

Other Notes

The project uses a CLI-based interface to keep it lightweight and accessible from terminals.

All dates and times are expected to be in the format:

YYYY-MM-DD HH:MM (e.g., 2025-08-01 09:30)

Productivity is calculated using simple metrics like total hours worked and deadlines met or missed.

CSV files should be well-formatted and include consistent headers.

References

Python Documentation: https://docs.python.org/3/

Python CSV Module:

https://docs.python.org/3/library/csv.html

Python datetime Module:

https://docs.python.org/3/library/datetime.html

Python argparse Module:

https://docs.python.org/3/library/argparse.html

Python unittest Framework:

https://docs.python.org/3/library/unittest.html