System Requirements Specification Index

For

Date and Time Processor

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Date and Time Processor

System Requirements Specification

1 Project Abstract

A healthcare scheduling system needs reliable date and time processing capabilities to manage patient appointments, medication schedules, and staff rotations. This assignment focuses on implementing fundamental datetime operations using Python's datetime module to create a practical scheduling tool. Students will learn how to perform essential datetime manipulations including conversions, calculations, formatting, and timezone handling for real-world scheduling scenarios.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. System must implement core datetime manipulation functions 2. Functions must handle date arithmetic, formatting, and timezone conversions 3. All functions require proper documentation (docstrings) 4. System must demonstrate practical applications of datetime processing 5. Error handling must validate inputs and handle edge cases

3 Constraints

3.1 Input Requirements

1. Date and Time Formats:
    - String dates must follow ISO format (YYYY-MM-DD)
    - String datetimes must follow format (YYYY-MM-DD HH:MM:SS)
    - Example: `"2025-03-19 14:30:00"`
2. Function Parameters:
    - Date strings must be validated before processing
    - Timezone values must be integer offsets (hours from UTC)
    - Duration inputs must be integers for days, hours, minutes
    - Date ranges must have valid start and end dates

3.2 Function Constraints

1. Function Definition:
    - Each function must have a specific datetime manipulation purpose
    - Must include docstrings with examples
    - Example: `def convert_string_to_datetime(date_string):`
2. Datetime Operations:
    - Must use Python's `datetime` module
    - Must handle string parsing and formatting
        - § `convert_string_to_datetime()`: Parse string to datetime object
        - § `format_datetime()`: Format datetime to specified string pattern

§ `calculate_date_difference()`: Find days between two dates

§ `add_time_duration()`: Add specified time to a datetime

§ `get_day_of_week()`: Return weekday name for given date

§ `convert_timezone()`: Convert datetime between timezone offsets

3. Return Values:
   - Functions must return appropriate types (datetime objects, strings, dictionaries)
   - The calculate_date_difference function must return a dictionary with different time units
   - Functions should handle standard use cases correctly

3.3 Output Constraints

1. Display Format:
   - Console output must be clearly labeled
   - Example: `Time difference: 7 days, 5 hours, 30 minutes`
   - Results should be formatted for readability
   - Main function should demonstrate each datetime operation with labeled results

4. Template code Structure:

1. Datetime Conversion Functions:

- `convert_string_to_datetime(date_string)` - converts string to datetime object
- `format_datetime(dt, format_string)` - formats datetime to specified string format

2. Datetime Calculation Functions:

- `calculate_date_difference(start_date, end_date)` - calculates time between dates
- `add_time_duration(dt, days=0, hours=0, minutes=0)` - adds time to datetime
- `get_day_of_week(date_string)` - returns weekday name for date

3. Timezone Functions:

   - `convert_timezone(dt, source_offset, target_offset)` - converts datetime between timezone offsets

4. Program Control:

   - `main()` - demonstrates all datetime functions with sample inputs
   - Should display results clearly with appropriate formatting

5. DETAILED FUNCTION IMPLEMENTATION GUIDE

5.1 String to Datetime Conversion Functions

Function: `convert_string_to_datetime(date_string)`

Requirements:

- Accept a string parameter in ISO format: "YYYY-MM-DD" or "YYYY-MM-DD HH:MM:SS"
- Use datetime.strptime() to parse the string with appropriate format patterns
- Try parsing with time format "%Y-%m-%d %H:%M:%S" first
- If that fails, try parsing date-only format "%Y-%m-%d"
- For date-only format, set time components to 00:00:00
- Return datetime.datetime object representing the parsed date

Examples:

- `convert_string_to_datetime("2025-03-19")` returns `datetime(2025, 3, 19, 0, 0)`
- `convert_string_to_datetime("2025-03-19 14:30:00")` returns `datetime(2025, 3, 19, 14, 30)`

5.2 Datetime Formatting Functions

Function: `format_datetime(dt, format_string="%Y-%m-%d %H:%M:%S")`

Requirements:

- Accept a datetime.datetime object as first parameter
- Accept optional format string parameter with default "%Y-%m-%d %H:%M:%S"
- Use datetime.strftime() method to format the datetime object
- Support standard strftime format codes (%Y, %m, %d, %H, %M, %S, %A, %B, etc.)
- Return formatted string representation of the datetime

Examples:

- `format_datetime(datetime(2025, 3, 19, 14, 30), "%B %d, %Y at %I:%M %p")` returns `"March 19, 2025 at 02:30 PM"`
- `format_datetime(datetime(2025, 3, 19), "%Y-%m-%d")` returns `"2025-03-19"`

5.3 Date Calculation Functions

Function: `calculate_date_difference(start_date, end_date)`

Requirements:

- Accept two parameters that can be either datetime objects or date strings
- Convert string inputs to datetime objects using convert_string_to_datetime()
- Calculate time difference using datetime subtraction (end_date - start_date)
- Handle both positive and negative differences (future and past dates)
- Use timedelta.total_seconds() to get precise time difference
- Calculate days using timedelta.days property
- Calculate total hours as total_seconds // 3600
- Calculate total minutes as total_seconds // 60
- Return dictionary with keys: "days", "hours", "minutes", "total_seconds"
- All values should be integers

Example:

- `calculate_date_difference("2025-03-19", "2025-03-26")` returns `{"days": 7, "hours": 168, "minutes": 10080, "total_seconds": 604800}`

Function: `add_time_duration(dt, days=0, hours=0, minutes=0)`

Requirements:

- Accept datetime object or string as first parameter
- Accept optional integer parameters for days, hours, and minutes (default 0)
- Convert string input to datetime object using convert_string_to_datetime()
- Support negative duration values for subtracting time
- Use timedelta(days=days, hours=hours, minutes=minutes) to create duration
- Add timedelta to datetime using + operator
- Return new datetime object with duration added

Examples:

- `add_time_duration(datetime(2025, 3, 19), days=2, hours=5)` returns `datetime(2025, 3, 21, 5, 0)`
- `add_time_duration("2025-03-19 10:00:00", days=-1, hours=-2)` returns `datetime(2025, 3, 18, 8, 0)`

5.4 Date Analysis Functions

Function: `get_day_of_week(date_string)`

Requirements:

- Accept either string date or datetime object as parameter
- Convert string input to datetime object using convert_string_to_datetime()
- Use strftime("%A") to get full weekday name in English
- Return string with full weekday name
- Return one of: "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"

Examples:

- `get_day_of_week("2025-03-19")` returns `"Wednesday"`
- `get_day_of_week(datetime(2025, 3, 19, 14, 30))` returns `"Wednesday"`

5.5 Timezone Conversion Functions

Function: `convert_timezone(dt, source_offset, target_offset)`

Requirements:

- Accept datetime object or string as first parameter
- Accept integer source timezone offset in hours
- Accept integer target timezone offset in hours
- Convert string input to datetime object using convert_string_to_datetime()

- Calculate hour difference: target_offset - source_offset
- Use timedelta(hours=hours_diff) to create time adjustment
- Add timedelta to original datetime to get converted time
- Return new datetime object in target timezone

Examples:

- `convert_timezone("2025-03-19 14:30:00", -5, -8)` returns `datetime(2025, 3, 19, 11, 30)` (Eastern to Pacific)
- `convert_timezone(datetime(2025, 3, 19, 23, 0), -5, 0)` returns `datetime(2025, 3, 20, 4, 0)` (crosses to next day)

5.6 Main Demonstration Function

Function: `main()`

Requirements:

- Print clear section headers for each demonstration
- Use sample data: appointment_date, surgery_date, staff_shift_start
- Demonstrate string to datetime conversion with clear before/after output
- Show datetime formatting with custom format patterns
- Calculate and display time differences between appointments
- Demonstrate duration addition for follow-up appointments
- Show day of week determination for multiple dates
- Demonstrate timezone conversion between Eastern (UTC-5) and Pacific (UTC-8)
- Format all output for readability with descriptive labels

Example output sections:

- "1. String to Datetime Conversion:"
- "2. Datetime Formatting:"
- "3. Date Difference Calculation:"
- "4. Time Duration Addition:"
- "5. Day of Week Determination:"
- "6. Timezone Conversion:"

5.7 Implementation Guidelines

Basic Requirements:

- Use Python's datetime module for all date and time operations
- Include proper docstrings for all functions with examples
- Handle both string and datetime inputs where specified
- Return appropriate data types as specified

- Focus on correct functionality for valid inputs
- Implement clean, readable code with proper formatting

Function Behavior:

- Functions should work correctly with properly formatted inputs
- String dates should be in ISO format (YYYY-MM-DD or YYYY-MM-DD HH:MM:SS)
- Timezone offsets should be reasonable integer values
- Duration parameters should be integers
- Functions should return the specified data types and structures

6. Execution Steps to follow:

1. Run the program

2. Observe how each datetime function processes date and time data

3. Test with sample inputs for string parsing, calculations, and formatting

4. Experiment with different datetime operations for scheduling scenarios

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

- This editor Auto Saves the code

- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

- To launch application: python3 filename.py

- To run Test cases: python3 -m unittest

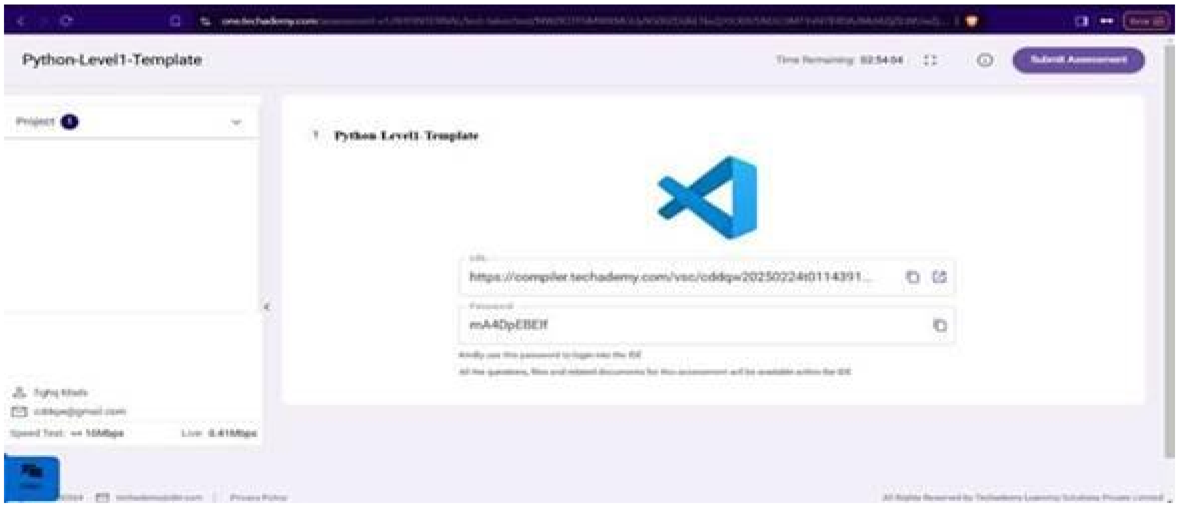
- Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

<u>Screen shot to run the program</u>

To run the application

Python3 filename.py

To run the testcase python -m unittest



Python-Level1-Template

Python Level1 Template

https://compiler.techademy.com/vsc/cddqw20250224t0114391...

mA4DpEBEf

- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.