Arash kheirollahi146731245 akheirollahi@mySeneca.ca  
note to professor: for some reason office word wouldn’t save my answers on the original file. I lost all my of my progress once :/

**Using Mozilla Firefox is strongly recommended for this Activity because it can transform JSON responses into a human readable format.** (Raw Data > "Pretty Print") (#)*is**points for API and Time Zone questions.*

1. (5) What is sent via the API from one system to another? What is sent back?

API sends: A request, typically in JSON or XML format, containing data or instructions.

Sent back: A response, usually in JSON or XML, with the requested data or a status message (success, error, etc.).

2. (5) Use api.agify.io to predict the age of a person using your given name and an ISO country code

API URL request: https://api.agify.io/?name=john&country\_id=US

JSON response: {

    "count": 96469,

    "name": "john",

    "age": 68,

    "country\_id": "US"

  }

3. (5) Use the time zone API request at worldtimeapi.org

API URL request: http://worldtimeapi.org/api/timezone/Asia/Tehran

JSON response: {

    "utc\_offset": "+03:30",

    "timezone": "Asia/Tehran",

    "day\_of\_week": 4,

    "day\_of\_year": 72,

    "datetime": "2025-03-13T01:51:23.547384+03:30",

    "utc\_datetime": "2025-03-12T22:21:23.547384+00:00",

    "unixtime": 1741818083,

    "raw\_offset": 12600,

    "week\_number": 11,

    "dst": false,

    "abbreviation": "+0330",

    "dst\_offset": 0,

    "dst\_from": null,

    "dst\_until": null,

    "client\_ip": "142.181.98.7"

}

4. (16)Using the above JSON data from worldtimeapi.org, fill in the JSON key / value pairs relating to the descriptions in the table below.

|  |  |  |
| --- | --- | --- |
| *See Response Schema* | JSON key | JSON value |
| UTC date/time in ISO8601 format | utc\_datetime | 2025-03-12T22:21:23.547384+00:00 |
| Unix UTC timestamp | unixtime | 1741818083 |
| Unix UTC to location difference | raw\_offset | 12600 (seconds) |
| Location's daylight-saving time difference | dst\_offset | 0 |
| Location date/time in ISO8601 format | datetime | 2025-03-13T01:51:23.547384+03:30 |
| How do you calculate the *location's* *timestamp* from the UTC timestamp using JSON keys? | *1741818083 (UTC)*  *+ 12600 (raw\_offset)*  *+ 0 (dst\_offset)*  *= 1741830683 (local timestamp)* | *Calculated location timestamp value is:*  1741830683 |

**5.**  (5) How did you confirm that your location timestamp when converted to data/time was the same as the Location date/time in ISO8601 format in the JSON schema? Show your test and the result.

Expected timestamp from JSON: "2025-03-13T01:51:23.547384+03:30"

Calculated timestamp: "2025-03-13T01:51:23+03:30"

**SDLC – Software Development Life Cycle 54 points = 9 points × 6 items, 75+ words each**

**Determine**:

The first step is understanding the problem statement given in the assignment. To become comfortable with the scope, I carefully read the assignment prompt multiple times. I highlight key requirements and constraints, ensuring I fully grasp the expected outcomes. I also review lecture notes and textbook references to find relevant concepts. If I still have uncertainties, I ask my professor or classmates for clarification. To create a plan, I break the task into smaller steps, estimating how much time each step will take and setting deadlines to complete them on time.

**Define:**

After understanding the problem, I define the detailed requirements. This includes identifying inputs (e.g., user data, files, or system arguments), the processing required (e.g., loops, conditionals, calculations), and expected outputs. I review the rubric and sample test cases to ensure I meet all requirements. Writing out sample inputs and expected outputs in a table helps me visualize how the program should function. If external libraries or specific programming functions are required, I document them early to avoid confusion during development.

**Design**:

Before coding, I outline a structured plan for implementation. I create pseudocode to map out the logic in a clear, step-by-step manner. If the problem is complex, I draw a flowchart to visualize the process. I also consider breaking the program into smaller functions for better modularity. Writing function headers and brief comments in advance helps ensure I stay focused while coding. This step ensures that my logic is sound before I write actual code, reducing errors later.

**Develop:**

With the design in place, I begin coding the solution. I write the program incrementally, testing each part before moving to the next. I use meaningful variable names and add comments to explain complex logic. When errors occur, I debug systematically by printing intermediate values and using debugging tools. I test with multiple cases, including edge cases, to ensure correctness. For example, in a recent assignment that required handling user input, I tested valid inputs, invalid inputs, and boundary conditions to verify robust error handling.

**Deliver**:

Once I am confident in my program, I prepare for submission. If required, I upload the project to the matrix server and verify that it runs correctly in the target environment. I check for formatting requirements and confirm that all files are included. If any unexpected issues arise during submission, I troubleshoot by reviewing error messages, revisiting instructions, or seeking assistance. Before finalizing, I reflect on the process by writing a short summary of what I learned and the challenges I faced, which helps improve my approach for future assignments.

**D'oh**:

Even after submission, maintaining the code is important. If the assignment is a stepping stone for future projects, I ensure my code is clean and well-documented for easy updates. I keep backups of my code using Git or cloud storage, making sure the latest version is always accessible. If additional requirements are added later, I update the program methodically instead of rewriting everything. In a recent assignment, I had to modify a function to accommodate new specifications, and because my code was modular and well-documented, making changes was straightforward.

**Software Version** **5 × 2 points each**

A. The software in focus is **Visual Studio Code**. As of February 2025, the latest stable release is **version 1.98**.

B. VS Code employs a versioning system in the format major.minor. Here's a breakdown:​

Major Version (1): Signifies the primary release of the software. Since its inception, VS Code has maintained the major version 1, indicating a consistent core platform.​

Minor Version (98): Represents incremental updates that include new features, improvements, and fixes. Each minor version reflects the ongoing enhancements made to the software.​

C. Forward compatibility refers to a software's ability to handle data or projects from future versions. While VS Code is designed to be extensible and adaptable, ensuring full forward compatibility can be challenging due to unforeseen changes in future versions. However, the development team strives to maintain a stable API, allowing extensions and configurations to function across future releases.

D. Backward compatibility ensures that newer versions of software can support projects or data from older versions. VS Code is generally backward compatible, allowing users to open and work on projects created in earlier versions without issues. This approach minimizes disruptions and ensures a seamless transition when updating to newer versions.

E. URL: Visual Studio Code February 2025 Release Notes​

web-cdn.bsky.app

Release Date: March 5, 2025​

web-cdn.bsky.app

gerardorenteria.blog

Description of One of the Latest Changes:

One notable enhancement in version 1.98 is the Next Edit Suggestions (Preview) feature. GitHub Copilot now predicts the next edit you are likely to make, streamlining the coding process and improving efficiency.​