Завдання №1

Завдання:

Розробити додаток для конвертації між одиницями відстані з підтримкою метричної та імперської систем вимірювання. Співвідношення для конвертації ви можете взяти з таблиці. Початково додаток повинен розпізнавати метри (*m*), сантиметри (*cm*), дюйми (*in*) та фути (*ft*), і підтримувати конвертацію між будь-якими з цих одиниць виміру.

Також необхідно реалізувати можливість розширювати список одиниць, що підтримуються, шляхом задання правил конвертації за допомогою JSON файлу. Формат JSON файлу - на ваш розсуд. Для прикладу, розширте ваш додаток додавши в файл значення для міліметрів (*mm*), ярдів (*yd*) та кілометрів (*km*).

Вхідні параметри:

Об'єкт у форматі JSON, що містить відстань для конвертації (*distance*) зі значенням (*value*) та шкалою (*unit*), а також позначення одиниці виміру для шкали, в яку повинна бути зроблена конвертація (*convertTo*), наприклад:

```
{"distance": {"unit": "m", "value": 0.5}, "convertTo": "ft"}
```

Вихідні дані:

Об'єкт у форматі JSON, що містить отримане значення відстані, округлене до сотих, а також позначення відповідної одиниці виміру, наприклад:

```
{"unit": "ft", "value": 1.64}
```

Завдання №2

Завдання:

Розробити простий додаток для сортування та відбору даних за визначеними правилами. Додаток повинен вміти працювати зі списками JSON об'єктів довільної структури, відбирати об'єкти, що містять ключі з відповідними значеннями, а також сортувати об'єкти за значенням, використовуючи природний порядок сортування.

Наприклад, якщо для даних виду:

```
{"data": [{"name": "John", "email": "john2@mail.com"},
          {"name": "John", "email": "john1@mail.com"},
          {"name": "Jane", "email": "jane@mail.com"}]}
```

задати умову:

```
{"condition": {"include": [{"name": "John"}], "sortBy": ["email"]}}
```

що містить два правила - *include* і *sortBy* (де правило *include* приймає набір пар ключ:значення для перевірки записів на відповідність, а правило *sortBy* приймає набір ключів для сортування), результатом буде об'єкт, що містить лише записи з ім'ям *John*, відсортовані по ключу *email*:

```
{"result": [{"name": "John", "email": "john1@mail.com"},
            {"name": "John", "email": "john2@mail.com"}]}
```

Окрім цього потрібно враховувати, що кожне правило фільтрації може мати декілька умов, наприклад правило побудоване за принципом “АБО”:

```
{"condition": {"exclude": [{"name":”John”},{"email":”john2@mail.com”}]}}
```

Тобто потрібно виключити користувачів які мають АБО "name":”John” АБО "email":”john2@mail.com”.

```
{"result": [{"name": "Jane", "email": "jane@mail.com"}]}
```

Іншим варіантом правила може бути правило побудоване за принципом “ТА”, наприклад:

```
{"exclude": [{"name": John, "email":”john2@mail.com”}]}}
```

Тобто потрібно виключити користувачів які мають "name":”John” ТА "email":”john2@mail.com”.

```
{"result": [{"name": "John", "email": "john1@mail.com"},
            {"name": "Jane", "email": "jane@mail.com"}]}
```

Плануючи підхід до дизайну коду додатка, необхідно передбачити можливість розширення функціонала шляхом додавання у код нових “модулів” з правилами. Важливо, щоб усі модулі мали між собою ідентичну структуру, були ізольовані один від одного та іншого коду додатка, та взаємодіяли з основним кодом за єдиним принципом. Для прикладу, ви можете додати новий модуль з правилом exclude, яке буде відкидати записи, що містять ключі з певним значенням.

Вхідні параметри:

JSON об'єкт зі списком даних (*data*), та умовою для обробки (*condition*):

```
{"data": [{"user": "mike@mail.com", "rating": 20, "disabled": false},
          {"user": "greg@mail.com", "rating": 14, "disabled": false},
          {"user": "john@mail.com", "rating": 25, "disabled": true}],
 "condition": {"exclude": [{"disabled": true}], "sortBy": ["rating"]}}
```

Вихідні дані:

JSON об'єкт з даними отриманими після застосування умови обробки (*result*):

```
{"result": [{"user": "greg@mail.com", "rating": 14, "disabled": false},
            {"user": "mike@mail.com", "rating": 20, "disabled": false}]}
```

Завдання №3 ‡

Завдання:

В деякій частині космосу є нерухомо розташований астероїд з унікальним мінералом. Для його точного знаходження був розроблений новий тип простих одноразових зондів, які під час активації один раз визначають точну відстань від себе до астероїда.

Необхідно написати функцію, яка задаватиме координати активації зондам і, отримуючи відстані від кожного з них до астероїда, знайде координати астероїда, витративши найменшу кількість зондів.

Для спрощення задачі припустимо, що частина космосу, в якій розташований рідкісний астероїд та можуть бути запущені зонди, обмежена уявним кубом розміром 100x100x100. А координати астероїда і зондів можуть бути лише цілими числами від 0 до 100.

Вхідні параметри:

Для вибору координат астероїда необхідно написати функцію, яка згенерує випадкове місцезнаходження астероїда *a(x, y, z)*. Також необхідно підготувати окрему функцію, яка, отримавши координати зонда, буде повертати відстань між ним та точкою *a*.

Вихідні дані:

Як результат програма повинна повернути координати астероїда *(location)*, кількість використаних зондів (*probes.count*) та їх координати *(probes.coordinates)*:

‡ *Ви можете вирішити тільки одне з завдань: або №3, або №4, вирішення обох завдань буде плюсом*

```
{"result": {
  "location": {"x":34, "y": 50, "z": 60},
  "probes": {
    "count": 44,
    "coordinates": [{"x": 10, "y": 9, "z": 21}, ..., {"x": 10, "y": 4, "z": 11}]
  }
}}
```

Завдання №4 ‡

Завдання:

Необхідно реалізувати опитувальник, в якому порядок та список запитань залежить від переданої конфігурації у форматі JSON. Опитувальник повинен підтримувати лише запитання з варіантами відповідей, наприклад:

```
{"What is your marital status?": ["Single", "Married"]}
{"Are you planning on getting married next year?": ["Yes", "No"]}
{"How long have you been married?": ["Less than a year", "More than a year"]}
{"Have you celebrated your one year anniversary?": ["Yes", "No"]}
```

Запитання в опитувальнику повинні визначатися динамічно на основі відповідей користувача - наступне запитання повинно залежати від відповіді на попереднє. Вам необхідно продумати, як буде працювати ця логіка, і розробити формат JSON конфігурації (він буде відрізнятись від прикладу вище), яка дозволить задавати правила, що пов'яжуть запитання з відповідями.

Для тестування роботи опитувальника потрібно створити скрипт, що працює з кодом логіки опитування і проходить по всім можливим шляхам опитувань. Для вирішення завдання не потрібно реалізовувати інтерфейс користувача, достатньо імплементувати скрипт та логіку опитування.

Вхідні параметри:

JSON конфігурація, у вибраному вами форматі, з запитаннями та доступними відповідями, що пов’язує відповіді користувача та наступні запитання.

Вихідні дані:

JSON об’єкт, що є результатом роботи скрипту тестування, з інформацією про кількість всіх можливих шляхів опитувань (*paths.number*), та всіма можливими послідовностями запитань з відповідями (*paths.list*):

‡ Ви можете вирішити тільки одне з завдань: або №3, або №4, вирішення обох завдань буде плюсом

```
{"paths": {"number": 3, "list": [
    [{"What is your marital status?": "Single"},
     {"Are you planning on getting married next year?": "Yes/No"}],
    [{"What is your marital status?": "Married"},
     {"How long have you been married?": "Less than a year"}],
    [{"What is your marital status?": "Married"},
     {"How long have you been married?": "More than a year"},
     {"Have you celebrated your one year anniversary?": "Yes/No"}],

]}}
```

Примітки до виконання завдань

Під час написання додатків, зверніть увагу на наступне:

- усі завдання повинні бути вирішені БЕЗ використання бібліотек чи фреймворків, наприклад таких як: React, Angular, Vue.js, Express
- файли додатків, що містять JavaScript або TypeScript код, обов'язково повинні мати відповідне розширення **.js*, або **.ts*
- фінальне рішення може бути орієнтоване як на запуск в браузері, так і в Node.js середовищі - на ваш розсуд
- код додатків необхідно розбити на логічні блоки, так щоб він був компактним, легко читався, та не містив повторень
- додатки повинні коректно реагувати на широкий спектр можливих вхідних значень, та обробляти виняткові ситуації
- всі завдання повинні бути вирішені оптимальним чином, з найменшим використанням ресурсів пам'яті та процесора

Виконане завдання (код) необхідно заархівувати в **.zip* архів (попередньо виключити з коду *node_modules,* якщо вони присутні), назвати архів по шаблону *`${surname}_${name}`* (наприклад Tarasenko_Taras.zip) та надіслати на email *hr@sysgears.com,* як тему листа вкажіть: “Виконані завдання. [Прізвище Ім'я]”.

Додатково, до листа необхідно прикріпити резюме в **.pdf* форматі, також назване за шаблоном *`${surname}_${name}`* (наприклад Tarasenko_Taras.pdf).