Задание №1

Задача:

Разработать приложение для конвертации между единицами измерения расстояния с поддержкой метрической и имперской систем мер. Соотношения для конвертации вы можете взять из таблицы. По умолчанию, приложение должно распознавать метры (*m*), сантиметры (*cm*), дюймы (*in*) и футы (*ft*), и поддерживать конвертацию между любыми единицами.

Также необходимо реализовать возможность расширять список поддерживаемых единиц путем задания правил конвертации посредством JSON файла. Формат JSON файла - на ваше усмотрение. В качестве примера, расширьте ваше приложение добавив в файл значения для миллиметров (*mm*), ярдов (*yd*) и километров (*km*).

Входящие параметры:

Объект в JSON формате, содержащий расстояние заданное для конвертации (*distance*) со значением (*value*) и шкалой (*unit*), а также обозначение единицы для шкалы, в которую должна быть произведена конвертация (*convert_to*). Например:

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

Выходные данные:

Объект в JSON формате, содержащий полученное значение расстояния, округленное до сотых, а также обозначение соответствующей единицы измерения, например:

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

Задание №2

Задача:

Разработать простое приложение для сортировки и отбора данных по заранее заданным правилам. Приложение должно уметь работать со списками JSON объектов произвольной структуры, отбирать объекты, содержащие ключи с определенными значениями, а также сортировать объекты по значениям, используя естественный порядок сортировки.

Например, если для данных вида:

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

задать условие:

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

содержащее два правила - *include* и *sort_by* (где правило *include* принимает набор пар ключ:значение для проверки записей на соответствие, а правило *sort_by* принимает набор ключей для сортировки), результатом будет объект, содержащий только записи с именем *John*, отсортированные по ключу *email*:

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

Планируя подход к дизайну кода приложения, необходимо предусмотреть возможность расширять функционал через добавление в код новых “модулей” с правилами. Важно, чтобы все модули имели между собой идентичную структуру, были изолированы друг от друга и остального кода приложения, и взаимодействовали с основным кодом, используя один и тот же подход. В качестве примера, вы можете добавить модуль с дополнительным правилом *exclude*, которое будет отбрасывать записи, содержащие ключи с определенным значением.

Входящие параметры:

JSON объект со списком данных (*data*), и условием для обработки (*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}], "sort_by": ["rating"]}}
```

Выходные данные:

JSON объект с данными полученными после применения условия обработки (*result*):

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

Задание №3 ‡

Задача:

Вам необходимо найти некоторую неизвестную, заранее заданную точку в трехмерном пространстве, за наименьшее количество попыток, полагаясь только на функцию, которая может возвращать расстояние от любой переданной вами в нее точки до искомой.

Для решения задачи сначала имплементируйте функцию *f*, которая, принимая координаты любой точки *s(x, y, z)*, возвращает расстояние между этой точкой и условно неизвестной, предварительно произвольно сгенерированной вами точкой *r(x, y, z)*, где *x*, *y*, и *z* могут быть любыми целыми числами между 0 и 100.

Например, для произвольно сгенерированной точки *r(0, 0, 10),* и переданной в функцию точки *s(0, 0, 0)*, результат работы функции будет следующим:

```
f(s) = 10  // расстояние между s(0, 0, 0) и r(0, 0, 10) равно 10
```

Далее реализуйте сам алгоритм решения задачи. Алгоритм должен находить координаты произвольно сгенерированной точки за наименьшее количество вызовов функции *f*. К решению приложите текстовое описание работы алгоритма.

Входящие параметры:

—

Выходные данные:

Координаты произвольно сгенерированной точки *r(x, y, z)*, координаты всех точек, переданных в функцию *f* вашим алгоритмом, а также количество вызовов функции *f*, за которое была найдена точка *r*.

```
{"result": {
    "random_point": {"x": 10, "y": 10, "z": 10},
    "search_points": [{"x": 0, "y": 1, "z": 2}, ..., {"x": 10, "y": 321, "z": 11}],
    "calls": 85
}}
```

‡ *Вы можете решить только одну из задач: или №3, или №4, решение обеих задач будет плюсом*

Задание №4 ‡

Задача:

Необходимо реализовать программируемый опросник, в котором порядок и список вопросов зависят от переданной конфигурации в JSON формате. Опросник должен поддерживать только вопросы с вариантами ответов, например:

```
{"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"]}
```

Вопросы в опроснике должны динамически определяться на основании ответов пользователя - следующий вопрос должен зависеть от ответа на предыдущий. Вам необходимо продумать, как будет работать эта логика, и разработать формат JSON конфигурации (он будет отличаться от примера выше), которая позволит задавать правила, связывающие вопросы с конкретными ответами.

Для тестирования работы опросника требуется создать скрипт, работающий с кодом логики опросника и проходящий по всем возможным путям опросов. Для решения задачи не требуется реализовывать пользовательский интерфейс, достаточно написать скрипт и логику опросника, последовательно предоставляющую вопрос и принимающую ответ.

Входящие параметры:

JSON конфигурация, в выбранном вами формате, с вопросами и допустимыми ответами, задающая связь между ответами пользователя и последующими вопросами.

Выходные данные:

JSON объект, являющийся результатом работы скрипта тестирования, с информацией о количестве всех возможных путей опросов (*paths.number*), и всеми возможными последовательностями вопросов с ответами (*paths.list*):

```
{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"}],

]}}
```

‡ *Вы можете решить только одну из задач: или №3, или №4, решение обеих задач будет плюсом*

Примечания к выполнению заданий

Во время написания программ, обратите внимание на следующее:

- все задания должны быть решены БЕЗ использования различных библиотек, или фреймворков, как например: React, Angular, Vue.js, или Express.js
- файлы программ, содержащие JavaScript и TypeScript код, обязательно должны иметь соответствующие расширения **.js*, или **.ts*
- итоговое решение может быть ориентировано как на запуск в браузере так и в среде Node.js - на ваше усмотрение
- код тестовых приложений необходимо разбить по логическим блокам, так чтобы он был компактным, легко читаемым, и не содержал повторений
- приложения должны корректно реагировать на широкий спектр возможных входных значений, и обрабатывать исключительные ситуации
- все задачи должны быть решены наиболее оптимальным образом, с наименьшим использованием ресурсов памяти и процессора

Выполненное задание (исходный код) необходимо заархивировать в **.zip* архив (предварительно исключить из кода *node_modules* если они присутствуют), назвать архив по шаблону `${name}_${surname}` (например taras_tarasenko) и отправить на email hr@sysgears.com, в качестве темы письма укажите: “Выполненные задания. [Имя Фамилия]”.

Дополнительно к письму необходимо прикрепить резюме в **.pdf* формате.