Маленький блог и бигдата к нему

Вы начали вести свой видеоблог. Он рассказывает про еду, технику или про что-то еще, что Вам так нравится.
Вам нужно понять, находит ли контент отклик у других, ведь видеоблогинг отнимает много сил. Поэтому Вы завели счётчик посетителей.
Вам нужно написать программу, которая поможет оценить посещаемость. В разные дни в блог может заходить разное количество людей, но главное, чтобы среднее количество посещений первое время не опускалось ниже 1000. Если посещаемость упадёт, нужно будет отправить уведомление.
Данные о посещаемости за первые три дня уже есть, и они записаны в переменные. Рекомендуемое среднее значение посещаемости хранится в переменной `expectedUsers`.
Теперь вам нужно добавить в программу расчет среднего количества посещений за наблюдаемый период. Если это значение окажется ниже рекомендуемого, Вам придётся задуматься о повышении посещаемости.
Среднее значение считается по формуле: сумма всех значений, поделенная на их количество. Давайте посчитаем среднее количество посещений за день и выведем это значение в консоль.

Задание

Рассчитаем среднюю посещаемость сайта.

1. Добавьте в конец программы объявление переменной `averageUsers`
2. И присвойте ей значение, посчитанное по формуле `(firstDayUsers + secondDayUsers + thirdDayUsers) / 3`
3. Затем выведите переменную `averageUsers` в консоль.

JS стартовый

```
let expectedUsers = 1000;

let firstDayUsers = 812;
let secondDayUsers = 1360;
let thirdDayUsers = 657;
```

Анализируем посещаемость

Программа работает, осталось сделать интерфейс более дружественным и добавить рекомендации.
Но сначала подпишите значение, которое выводится в консоль. Каждый должен видеть, что это среднее значение, а не что-то другое.
Потом нужно сравнить среднее значение с ожидаемым и показать рекомендацию. Если посещаемость хорошая (среднее значение больше ожидаемого), то подбодрите себя, если нет — попросите поднапрячься и начать готовить более качественный контент.

Задание

Завершим программу.

1. Добавьте подсказку к выводу среднего значения в консоль: `'Средняя посещаемость: ' + averageUsers`
2. В конец программы добавьте проверку того, что среднее значение больше ожидаемого.
3. Если условие выполняется, выведите в консоль сообщение `'Посещаемость великолепна. Продолжай в том же духе!'`
4. В противном случае, выведите в консоль сообщение `'Посещаемость так себе. Нужно поднапрячься!'`

Новые данные

Итак, Вас задела рекомендация поднапрячься с блогом. Тем более, что вчера ты выпустил крайне удачное новое видео. Услышав про недостаток данных, ты временно успокоился. Но сразу же добавил данные за вчера и требуется вставить их в программу. Это нам на руку. Ведь успешное видео — это хорошая посещаемость. А это положительно отразится на средней посещаемости.
Ваша срочная задача — внести в программу новые данные. Для этого нужно не только добавить переменную с посещаемостью за вчера, но и обновить формулу расчёта среднего. В формуле нужно добавить ещё одно слагаемое в сумму, а также увеличить делитель с трёх до четырёх.

Задание

1. В следующей строке после переменной `thirdDayUsers` добавьте переменную `fourthDayUsers` со значением `1247`
2. Измените расчёт среднего значения. Добавьте `fourthDayUsers` последним слагаемым и измените делитель на `4`

Масса данных? Массив!

Удивительно, но видеоблог смог просуществовать достаточно долго, и пришли данные за последний месяц:

817, 1370, 752, 1247, 681, 1120, 915, 1281, 875, 1341, 757, 610, 812, 1170, 769, 1261, 845, 1289, 515, 1247, 845, 1311, 741, 1239, 812, 638, 877, 1242, 1159, 1372

Чтобы проанализировать эти данные, придётся добавить в текущую программу ещё 26 переменных. Много писанины? А скоро данные и за год подоспеют.
Ваша программа работает правильно, но дополнять её новыми данными крайне трудоёмко. Поэтому пришло время рефакторинга. Рефакторинг — это переписывание программы, после которого она должна работать так же, но быть более гибкой. Цель нашего рефакторинга — снизить трудоёмкость добавления и изменения массивного количества данных.
Благо, существует способ хранить массивные данные. Этот способ так и называется — массив. Массив — это тип данных, который представляет собой список элементов, у каждого из которых есть свой порядковый номер.
Массивы создаются с помощью так называемого литерала массива — квадратных скобок. Внутри скобок через запятую перечисляются все значения, которые должен содержать массив. При создании массив, как любое значение, можно записать в переменную:

```
let numbers = [1, 2, 3, 4, 5];
```

Вообще, в массиве можно хранить любые данные: строки, булевы значения, числа и даже другие массивы.
Рефакторинг мы будем делать плавно, не удаляя всю программу, а переписывая кусками. На первом этапе избавимся от отдельных переменных для данных:

1. Временно присвоим среднему значению нулевое значение.
2. Создадим массив, в который перенесём значения из переменных.
3. И теперь, когда переменные для данных в программе больше не используются, смело их удаляем.

Задание

1. Вместо расчёта среднего значения присвойте переменной `averageUsers` значение 0
2. После объявленных переменных добавьте переменную `usersByDay` со значением `[812, 1360, 657, 1247]`
3. Смело удаляйте переменные `firstDayUsers`, `secondDayUsers`, `thirdDayUsers` и `fourthDayUsers`

Чтение из массива по индексу

Мы продолжим рефакторинг чуть позже. А сейчас поучимся работать с массивами. Массив — это цельный список, поэтому его можно записать в одну переменную и эту переменную передать в какую-нибудь команду. Согласитесь, что это удобнее, чем передавать в команду множество отдельных переменных.
При обработке массивов почти всегда нужно получать его отдельные значения. Чтобы получить элемент массива, нужно в уже знакомых квадратных скобках указать порядковый номер или индекс этого элемента.

```
let numbers = [1, 2, 3, 4, 5];
console.log(numbers[1]); // Выведет: 2
```

Нумерация элементов в массиве начинается с нуля: первый элемент массива идёт под номером ноль, второй — под номером один, третий — два и так далее.
Если вам трудно запомнить такой необычный способ нумерации, вы можете использовать аналогию с годами: летосчисление и возраст тоже начинаются с нуля. Если человеку 25 лет, это значит, что он живёт свой 26 год. А Михаэль Шумахер с 2000 по 2004 год выиграл пять чемпионских титулов подряд.
Подытожим. Чтобы получить первый элемент массива, например, `usersByDay`, нужно написать `usersByDay[0]`. Название переменной показывает, что мы обращаемся к массиву. Квадратные скобки говорят о том, что нам нужен не весь массив, а только один из его элементов. В квадратные скобки мы передаём номер нужного нам элемента.
Давайте попробуем прочитать ещё несколько записей из массива `usersByDay`.

Задание

Получим значения из массива.

1. После массива объявите переменную `firstDayUsers` и запишите в неё первый элемент массива: `usersByDay[0]`
2. В следующей строке выведите переменную `firstDayUsers` в консоль.
3. В следующей строке объявите переменную `fourthDayUsers` и запишите в неё четвёртый элемент массива: `usersByDay[3]`
4. В следующей строке выведите переменную `fourthDayUsers` в консоль.

JS стартовый

```
let usersByDay = [812, 1360, 657, 1247];
```

Переменная как индекс

В квадратные скобки можно передавать не только числа. Туда можно передать любое выражение. Это выражение в итоге будет преобразовано в число.
Давайте попрактикуемся с таким способом обращения к элементам массива и поочерёдно переберём их с помощью одной и той же переменной. Для этого нужно завести переменную для хранения текущего индекса элемента массива. Назовём её index и запишем в неё начальное значение ноль.
Если теперь вывести в консоль выражение `usersByDay[index]`, то мы увидим первое значение массива. Ведь сейчас в переменной хранится ноль, и выражение `usersByDay[index]` равнозначно `usersByDay[0]`.
Если изменить значение переменной `index` с нуля на единицу и снова обратиться к `usersByDay[index]`, то мы получим элемент с индексом 1, то есть второй элемент.
Смотрите, выражение в скобках в обоих случаях одинаковое: `usersByDay[index]`. Но в разных условиях оно возвращает разные значения.
Если мы продолжим увеличивать `index` и читать элементы массива `usersByDay`, то рано или поздно мы пройдём по всем элементам.

Задание

Получим элементы массива, изменяя переменную-индекс «вручную».

1. Удалите все строки, кроме объявления массива `usersByDay`
2. После массива объявите переменную `index` с нулевым значением.
3. На следующей строчке выведите в консоль `usersByDay[index]`
4. На следующей строчке задайте переменной `index` значение `3`
5. На следующей строчке снова выведите в консоль `usersByDay[index]`

Длина массива

Уже догадались, к чему мы идём? К перебору массивов в цикле!
Начальный индекс у массива мы знаем. Он равен нулю. Индексы в массиве возрастают на единицу. Использовать переменную в качестве индекса можно. Получается, что мы можем задать переменной нулевое значение, а затем использовать её в цикле для доступа к элементам массива, наращивая в каждой итерации на единицу.
Остался один вопрос: когда остановить цикл? Для этого надо знать длину массива.
И здесь тоже всё отлично. Массивы умеют рассказывать о своей длине. Для этого используется команда `[].length`, которая знает о количестве элементов в массиве:

```
let numbers = [1, 2, 3, 4];
console.log(numbers.length); // Выведет: 4
```

С помощью обращения к `length` можно получить последний элемент массива, даже если вы не знаете, сколько элементов в нём хранится:

```
someBigArray[someBigArray.length - 1];
```

Разберём это выражение. Допустим, в этом большом массиве 100 элементов. Его длина будет равна 100. Индекс последнего, сотого, элемента будет равен 99, потому что нумерация в массивах начинается с нуля. Поэтому индекс последнего элемента вычисляется вычитанием единицы из длины.
С помощью команды `[].length` вы можете единообразно работать с любыми массивами. То есть можно использовать один и тот же код для вычислений на массивах произвольной длины.

Задание

1. Снова удалите все строки, кроме объявления массива `usersByDay`
2. Во второй строке выведите в консоль длину массива `usersByDay`
3. На следующей строке выведите в консоль последний элемент массива: `usersByDay[usersByDay.length - 1]`
4. На следующей строке выведите в консоль третий с конца элемент массива, используя квадратные скобки и `length`
5. Теперь в первой строке, через запятую, добавьте в массив `usersByDay` пятый элемент `1000` и убедитесь, что вывод в консоль работает правильно.

Ударим циклами по массивам!

Циклы и массивы тесно связаны между собой. Для многих операций на массивах, например, подсчёта суммы элементов, используются циклы. Теперь и вы готовы использовать цикл для обхода массива.
Для этой задачи лучше всего подойдёт цикл `for`. Переменная цикла будет использоваться как индекс элементов массива. Поэтому зададим ей нулевое значение и будем увеличивать в каждой итерации на единицу, пока её значение не станет равным индексу последнего элемента.
Традиционное название переменной цикла `i` — это сокращение от `index`, то есть индекса или порядкового номера. Это тоже одно из соглашений среди программистов, которое позволяет сократить код.
Ещё один стилистический момент. Как лучше писать условие выхода из цикла? Есть два варианта. Рассмотрим их на примере массива из трёх элементов:

```
// Первый вариант: i < usersByDay.length
// usersByDay.length == 3

Подготовка: i = 0
1 итерация: i = 0; 0 < 3? да! действия первой итерации; i = 1
2 итерация: i = 1; 1 < 3? да! действия второй итерации; i = 2
3 итерация: i = 2; 2 < 3? да! действия третьей итерации; i = 3
4 итерация: i = 3; 3 < 3? нет! Завершаем цикл!
// Второй вариант: i <= usersByDay.length - 1
// usersByDay.length - 1 == 2

Подготовка: i = 0
1 итерация: i = 0; 0 <= 2? да! действия первой итерации; i = 1
2 итерация: i = 1; 1 <= 2? да! действия второй итерации; i = 2
3 итерация: i = 2; 2 <= 2? да! действия третьей итерации; i = 3
4 итерация: i = 3; 3 <= 2? нет! Завершаем цикл!
```

Оба варианта условия выхода, `i < usersByDay.length` и `i <= usersByDay.length - 1`, работают одинаково. Но пока мы будем использовать второй вариант, с вычитанием единицы. Он будет напоминать нам про непривычную нумерацию элементов массива, которая начинается с нуля.

Задание

1. После объявления массива добавьте цикл, который увеличивает переменную i с нуля до `usersByDay.length - 1` включительно. Значение i должно увеличиваться на единицу после каждой итерации.
2. Внутри цикла выведите в консоль значение `usersByDay[i]`

JS на момент задания

```
let expectedUsers = 1000;
```

```
let usersByDay = [812, 1360, 657, 1247];
// Рассчитываем среднее значение посещаемости
let averageUsers = 0;
console.log('Средняя посещаемость: ' + averageUsers);
if (averageUsers > expectedUsers) {
  console.log('Посещаемость великолепна. Продолжай в том же
духе!');
} else {
  console.log('Посещаемость так себе. Нужно поднапрячься!');
}
```

Суммирование в цикле

В цикле можно не только выводить элементы массива в консоль, но и проводить с ними какие-то операции. Например, суммировать. Для этого нужно перед циклом завести переменную с нулевым значением. Затем на каждой итерации прибавлять к ней значение очередного элемента массива. В итоге после цикла в переменной окажется сумма всех элементов.
Наша задача — найти среднее значение. А для этого сначала нужно узнать сумму всех элементов.
Давайте перепишем цикл так, чтобы он не выводил значения элементов в консоль, а последовательно складывал их между собой и записывал в одну переменную.

Задание

1. После объявления массива добавьте переменную `totalUsers` с нулевым значением;
2. Далее добавьте цикл, который увеличивает переменную `i` с нуля до `usersByDay.length - 1` включительно. Значение `i` должно увеличиваться на единицу после каждой итерации.
3. Внутри цикла выведите в консоль значение `usersByDay[i]`

Завершаем рефакторинг

На первом шаге рефакторинга мы избавлялись от отдельных переменных и намеренно отключили расчёт среднего значения, записав ноль в переменную `averageUsers`.
Пришло время «починить» этот механизм, но уже с использованием массива.
Вспоминаем, что среднее — это сумма всех элементов, делённая на их количество.
Сумму в цикле мы уже посчитали, а количество элементов можем получить, используя `[].length`.
Рефакторинг завершён, а вот и вишенка на торте! Теперь программа стала намного гибче и позволяет легко анализировать любые объёмы данных (за неделю, месяц, да хоть за год). Для этого нужно просто менять значения внутри массива `usersByDay`.
И, наконец, выполним первоочередную задачу. Проанализируйте посещаемость за последний месяц. Для этого нужно скопировать эти данные внутрь массива:

817, 1370, 752, 1247, 681, 1120, 915, 1281, 875, 1341, 757, 610, 812, 1170, 769, 1261, 845, 1289, 515, 1247, 845, 1311, 741, 1239, 812, 638, 877, 1242, 1159, 1372

Теперь программа универсальна и работает с массивами любой длины. Но есть один нюанс. Если запустить программу на пустом массиве, то средняя посещаемость получится NaN (не число). Оно и понятно, ведь в формуле подсчёта среднего и делителем, и делимым будет ноль, а 0 / 0 даёт неопределённый результат, то есть NaN. Если задуматься, это логично: мы не можем знать среднее значение несуществующих значений, поэтому можно считать, что наш алгоритм работает как нужно.

Задание

1. Удалите вывод в консоль переменной `totalUsers`
2. Задайте переменной `averageUsers` значение `totalUsers / usersByDay.length` вместо нуля.
3. Замените данные внутри массива `usersByDay` новыми данными. Скопируйте их из теории выше и вставьте между квадратными скобками.

JS after

```
let expectedUsers = 1000;

let usersByDay = [817, 1370, 752, 1247, 681, 1120, 915, 1281, 875, 1341, 757, 610, 812, 1170, 769, 1261, 845, 1289, 515, 1247, 845, 1311, 741, 1239, 812, 638, 877, 1242, 1159, 1372];

// Суммируем посещаемость
let totalUsers = 0;
for (let i = 0; i <= usersByDay.length - 1; i++) {
  totalUsers += usersByDay[i];
}
```

```
// Рассчитываем среднее значение посещаемости
let averageUsers = totalUsers / usersByDay.length;
console.log('Средняя посещаемость: ' + averageUsers);

if (averageUsers > expectedUsers) {
  console.log('Посещаемость великолепна. Продолжай в том же духе!');
} else {
  console.log('Посещаемость так себе. Нужно поднапрячься!');
}
```

Одно маленькое аналитическое расследование

Теперь надо покопаться в данных поглубже. Может и найдём проблему. А что если выделить дни с самыми большими провалами посещаемости? Например те дни, в которых посещаемость на 100 или даже 200 человек меньше, чем ожидаемая. Сделаем величину провала настраиваемой.
Сейчас проверим гипотезу. Вводим переменную с минимальной допустимой посещаемостью. Потом в цикле проверяем посещаемость за текущий день, и, если она не дотягивает до минимальной, выводим её в консоль. Интересно, много ли будет таких плохих значений?

Задание

1. После переменной `totalUsers` добавьте переменную `minUsers` со значением `expectedUsers - 100`
2. Внутри цикла добавьте проверку, что `usersByDay[i]` меньше `minUsers`
3. Если проверка проходит, выводите в консоль количество посетителей текущего дня.

Запись в массив по индексу

Вроде работает! Провальных дней много. Но пока ничего не понятно.
Предлагаю для прояснения картины собрать данные о провалах в новый массив. Для этого придётся записывать данные в массив. Запись в массив происходит так же, как и чтение — через обращение к элементу с помощью квадратных скобок:

```
let numbers = [];
let index = 1;
numbers[0] = 1;
numbers[index] = 2;
console.log(numbers); // Выведет: [1,2]
```

Обратите внимание, что если в массиве нет элемента под тем номером, под которым мы записываем, то этот элемент будет создан. До выполнения кода в массиве не было ни нулевого, ни первого элементов, но после того, как мы записали значения в эти позиции, элементы добавились в массив.
Чтобы сохранить согласованность с первым графиком, мы будем формировать массив с провальными днями в том же цикле, в котором суммируем посещаемость. Если посещаемость в `i`-й день хорошая, то будем записывать в `i`-й элемент массива провалов ноль, если посещаемость плохая, будем записывать туда величину провала. Чтобы посчитать величину провала, будем из ожидаемого значения вычитать фактическое значение посещаемости. Например:

- Посещаемость в пятый день составила 681 человек.
- Разность ожидаемой посещаемости и фактической: 1000 - 681 = 319.
- Значит, 319 человек — величина провала посещаемости в пятый день.

Задание

1. После переменной `minUsers` добавьте пустой массив `badDays`
2. Если проверка на дневной провал в посещаемости выполняется, записывайте в `i`-й элемент массива `badDays` значение `expectedUsers - usersByDay[i]`. Вывод в консоль текущего значения удалите.
3. Если условие не выполняется, записывайте в `i`-й элемент массива `badDays` ноль.

JS after

```
let expectedUsers = 1000;

let usersByDay = [817, 1370, 752, 1247, 681, 1120, 915, 1281, 875, 1341, 757, 610, 812, 1170, 769, 1261, 845, 1289, 515, 1247, 845, 1311, 741, 1239, 812, 638, 877, 1242, 1159, 1372];

// Суммируем посещаемость и анализируем провалы
let totalUsers = 0;
let minUsers = expectedUsers - 100;
```

```
let badDays = [];

for (let i = 0; i <= usersByDay.length - 1; i++) {
  totalUsers += usersByDay[i];
  if (usersByDay[i] < minUsers) {
    badDays[i] = expectedUsers - usersByDay[i];
  } else {
    badDays[i] = 0;
    }
}

// Рассчитываем среднее значение посещаемости
let averageUsers = totalUsers / usersByDay.length;
console.log('Средняя посещаемость: ' + averageUsers);

if (averageUsers > expectedUsers) {
  console.log('Посещаемость великолепна. Продолжай в том же духе!');
} else {
  console.log('Посещаемость так себе. Нужно поднапрячься!');
}
```

И, наконец, выводы!

Картинка начала складываться.
Теперь поэкспериментируем с порогом фильтрации плохих дней. Возможно, получится выявить какую-то закономерность.
За порог чувствительности фильтра плохих дней отвечает переменная `minUsers`.
Сейчас плохими считаются дни с посещаемостью на 100 человек ниже ожидаемой.
Меняя размер вычитаемого значения, можно настраивать чувствительность.

```
// Чувствительность ниже, плохих дней в массиве больше
let minUsers = expectedUsers - 100;

// Чувствительность выше, плохих дней в массиве меньше
let minUsers = expectedUsers - 200;
```

Теперь попробуйте что-нибудь накопать

Задание

Настроим фильтр плохих дней, чтобы найти истинную причину проблемы.

1. Снизьте минимально допустимое количество пользователей ещё на 100 человек.
2. И ещё на 100!

Конспект «Массивы». Раздел 1

Массив — тип данных, который представляет собой список элементов, у каждого из которых есть свой порядковый номер.
В массиве можно хранить любые данные: строки, булевы значения, числа и даже другие массивы.
Нумерация элементов массива начинается с нуля, поэтому порядковый номер (индекс) первого элемента равен нулю.
В качестве индекса можно использовать переменную.
Используйте команду [].length, чтобы узнать длину массива (сколько в нём элементов). С её помощью можно получить последний элемент массива.

```
let numbers = [1, 2, 3, 4, 5];
let index = 3;

console.log(numbers[0]);                  // Выведет: 1
console.log(numbers[index]);              // Выведет: 4
console.log(numbers.length);              // Выведет: 5
console.log(numbers[numbers.length - 1]); // Выведет: 5
```

Массивы можно перебирать в циклах. Например, цикл ниже выводит элементы массива в консоль по очереди и прекращает свою работу, когда i станет равно длине массива.

```
let numbers = [1, 2, 3, 4, 5];

for (let i = 0; i <= numbers.length - 1; i++) {
  console.log(numbers[i]);
}

// Выведет: 1
// Выведет: 2
// Выведет: 3
// Выведет: 4
// Выведет: 5
```

Запись в массив происходит так же, как чтение, через квадратные скобки.

```
let numbers = [];
let index = 1;

numbers[0] = 1;
numbers[index] = 2;

console.log(numbers); // Выведет: [1,2]
```

Шифрование. Home Edition

Многие из вас в раннем возрасте придумывали свои секретные шифры, кодированные сообщения и даже алфавиты. У многих это осталось забавой детства, но для специалистов криптографии - это очень серьезная работа. От этого зависит конфиденциальности наших данных. Поэтому сегодня нам предстоит создать простейший дешифратор.
Есть массив symbols, в котором хранится алфавит (буквы и другие символы).
Есть массив encodedSymbols, в котором хранится зашифрованное сообщение. Каждый элемент этого массива — это индекс символа из массива symbols.
Программа дешифровки должна переводить элементы из массива с шифровкой (encodedSymbols) в символы из массива алфавита (symbols) и склеивать из них расшифрованную строку. Эту строку запиши в переменную decodedMessage

JS стартовый

```
// Алфавит
let symbols = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И',
'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', 'Р', 'С', 'Т', 'У', 'Ф', 'Х',
'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я', 'а', 'б', 'в',
'г', 'д', 'е', 'ё', 'ж', 'з', 'и', 'й', 'к', 'л', 'м', 'н', 'о',
'п', 'р', 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ъ', 'ы',
'ь', 'э', 'ю', 'я', ' ', '.', ',', '—', '!'];

// Закодированное сообщение
let encodedSymbols = [18, 38, 46, 62, 66, 50, 33, 41, 66, 49, 48,
38, 58, 62, 68, 66, 48, 37, 42, 47, 66, 50, 33, 41, 66, 49, 48,
51, 49, 42, 67];

// Раскодированное сообщение
let decodedMessage = '';
```

Подозрения

Допустим, посещаемость уже несколько месяцев держится выше запланированной. Но популярность видеоблога растёт подозрительно медленно: число лайков, шеров, подписчиков явно не соответствует посещаемости.
Вы заподозрили неладное и решили снова проанализировать данные за последний месяц. Вот они:

```
817, 581, 1370, 752, 1247, 681, 1120, 915, 875, 1341, 757, 610,
812, 741, 1139, 812, 638, 877, 1242, 1159, 1372, 1170, 845, 1289,
515, 1247, 769, 1261, 2805, 1201
```

Задание

1. Замените устаревшие данные новыми и проверьте, есть ли подозрительные провалы посещаемости.

JS стартовый

```
let expectedUsers = 1000;

let usersByDay = [817, 581, 1370, 752, 1247, 681, 1120, 915, 875,
1341, 757, 610, 812, 741, 1139, 812, 638, 877, 1242, 1159, 1372,
1170, 845, 1289, 515, 1247, 769, 1261, 2805, 1201];

// Суммируем посещаемость и анализируем провалы
let totalUsers = 0;
let minUsers = expectedUsers - 300;
let badDays = [];

for (let i = 0; i <= usersByDay.length - 1; i++) {
  totalUsers += usersByDay[i];
  if (usersByDay[i] < minUsers) {
    badDays[i] = expectedUsers - usersByDay[i];
  } else {
    badDays[i] = 0;
  }
}

// Рассчитываем среднее значение посещаемости
let averageUsers = totalUsers / usersByDay.length;
console.log('Средняя посещаемость: ' + averageUsers);

if (averageUsers > expectedUsers) {
  console.log('Посещаемость великолепна. Продолжай в том же
духе!');
} else {
  console.log('Посещаемость так себе. Нужно поднапрячься!');
```

```
}
```

Меняем элементы местами

Опасения подтверждаются: в посещаемости стало даже больше сильных провалов, чем раньше. Анализатор всегда говорит, что средняя посещаемость хорошая, и никто даже не думает искать там провалы. Что же происходит с программой? Посмотрите на всплеск посещаемости в конце месяца. Это маркетологи нагоняют некачественный трафик. Этот пик, в свою очередь, влияет на среднюю посещаемость, и она оказывается хорошей.

Придётся улучшить программу и, помимо средней посещаемости, считать медианную посещаемость, которая менее чувствительна к подобным всплескам. Медиана — это срединное значение массива. Чтобы его получить, вначале нужно отсортировать массив. Поэтому сейчас будем учиться делать сортировку.
При сортировке элементы массива меняются местами. Обычно это делается через промежуточную переменную, в которую сохраняется один из переставляемых элементов. Попробуйте переставить два элемента сами.

Задание

Поменяем местами первый и второй элементы массива.

1. В конец программы добавьте переменную swap со значением `usersByDay[0]`
2. Затем в элемент массива с индексом 0 запишите значение элемента с индексом 1
3. Ниже выведите массив в консоль. Смотрите, первое значение потеряется, если мы не сохраним его в переменную раньше.
4. Затем в элемент массива с индексом 1 запишите значение переменой `swap`
5. И ещё раз выведите массив в консоль.

JS стартовый

```
let usersByDay = [4, 1, 2, 3];
console.log(usersByDay);
```

Ищем минимальный элемент

С перестановкой элементов попрактиковались. Подытожим, зачем нужна вспомогательная переменная. Допустим, есть массив, в котором мы меняем местами первый и второй элементы без вспомогательной переменной:

```
let numbers = [1, 2, 3];
// Теперь numbers такой [2, 2, 3]
numbers[0] = numbers[1];
```

Если сразу записать на первое место значение второго элемента, то мы потеряем значение первого элемента. Поэтому сначала нужно значение первого элемента сохранить в переменную:

```
let numbers = [1, 2, 3];
// Теперь в swap хранится 1
let swap = numbers[0];
// Теперь numbers такой [2, 2, 3]
numbers[0] = numbers[1];
// Теперь numbers такой [2, 1, 3]
numbers[1] = swap;
```

Следующий шаг на пути к сортировке — поиск минимального элемента. И этот элемент мы будем искать не во всём массиве, а в указанной его части.
Для этого заведём переменную `currentIndex`. Она будет управлять начальным значением переменной цикла. Обратите внимание, что переменная цикла в этот раз будет называться `j` (это ещё одно типовое название)

Задание

1. Оставьте только первые две строки кода
2. В третью добавьте переменную `currentIndex` с нулевым значением.
3. Затем добавьте цикл, который увеличивает переменную `j` со значения `currentIndex + 1` до `usersByDay.length - 1` включительно. Значение `j` должно увеличиваться на единицу после каждой итерации.
4. Внутри цикла выводите в консоль значение `j`-го элемента массива.

Минимальный элемент найден!

Почему начальное значение переменной цикла задано как `currentIndex + 1`? Это нужно, чтобы искать минимальное значение после элемента с позицией `currentIndex`.

```
let currentIndex = 0;
// Минимальный элемент ищем начиная со второй позиции
// [4, 2, 1, 3]
let currentIndex = 1;
// Минимальный элемент ищем начиная с третьей позиции
// [4, 2, 1, 3]
```

Как найти минимальный элемент, расположенный после первого?

1. Добавим переменную `minValue` для хранения минимального значения.
2. Предположим, что первый элемент и есть минимальный. Поэтому до цикла сохраним в `minValue` значение первого элемента.
3. На каждой итерации цикла сравниваем текущий элемент со значением `minValue`.
4. Если текущий элемент меньше `minValue`, то записываем его в `minValue`.

Если первый элемент и был минимальный, то в цикле значение `minValue` не изменится, если же после первого элемента были элементы с меньшим значением, то это значение запишется в `minValue` в цикле.
В любом случае минимальный элемент будет найден.
Этот алгоритм не будет работать на пустом массиве. Чтобы алгоритм работал, нам нужен, как минимум, один элемент, чтобы обозначить его как минимальный до начала работы цикла.

Задание

1. После переменной currentIndex добавьте переменную `minValue` со значением `usersByDay[currentIndex]`
2. Затем в цикле удалите вывод в консоль и добавьте условие `usersByDay[j] < minValue`
3. При выполнении условия присвойте переменной `minValue` значение `j`-го элемента массива.
4. И там же выведите в консоль сообщение: `'Новый минимальный элемент: ' + minValue`
5. После цикла выведите в консоль сообщение: `'Минимальный элемент: ' + minValue`

Начинаем сортировку

Мы научились находить минимальное значение. Почему бы не сделать ещё шаг. Теперь давайте не просто находить минимальное значение после первого элемента, а записывать это значение на место первого элемента. Для этого нужно немного дополнить алгоритм:

1. Добавим переменную `minValue` для хранения минимального значения.
2. Предположим, что первый элемент и есть минимальный. Поэтому до цикла сохраним в `minValue` значение первого элемента.
3. На каждой итерации цикла сравниваем текущий элемент со значением `minValue`.
4. Если текущий элемент меньше `minValue`, то записываем его в `minValue`, а затем меняем местами значение первого элемента и текущего.

После завершения работы цикла на первой позиции массива окажется элемент с минимальным значением. Все остальные элементы будут больше.

Задание

1. Внутри условия после переменной `minValue` сохраните значение `usersByDay[currentIndex]` в переменную `swap`. Оставшийся код тоже пишите внутри условия.
2. На следующей строке в `usersByDay[currentIndex]` запишите минимальное значение.
3. Ниже запишите в `usersByDay[j]` значение переменной `swap`
4. На следующей строке выведите в консоль: `'Меняю местами ' + swap + ' и ' + minValue`
5. Ниже выведите в консоль: `'Массив сейчас: ' + usersByDay`

Продолжаем сортировку

Вы прошлись по всему массиву, нашли минимальный элемент и поместили его на первое место.
Как продолжить сортировку? Очень просто. Теперь нужно повторить всё то же самое со вторым элементом массива: ищем минимальные элементы в оставшейся части массива и помещаем их на второе место.
После этого шага на первой позиции массива окажется самый маленький элемент, на второй позиции — следующий по величине. А на остальных позициях — элементы с более крупными значениями.
Затем нужно будет повторить эти же действия с третьим элементом.
После этого массив будет отсортирован полностью, так как на первых трёх позициях по возрастанию будут размещены минимальные элементы, а на четвёртой позиции автоматически останется максимальный элемент.
Реализуем эти шаги через вставку и копирование кода. Единственное, что будет отличаться в разных кусках — это значение переменной `currentIndex`.

Задание

6. В конце программы вставьте первый блок кода.
7. В раскомментированном коде измените значение `currentIndex` на `1`
8. Раскомментируйте второй блок кода.
9. В раскомментированном коде измените значение `currentIndex` на `2`

Первый блок

```
/*
console.log(usersByDay);
currentIndex = 0;
minValue = usersByDay[currentIndex];

for (let j = currentIndex + 1; j <= usersByDay.length - 1; j++) {
  if (usersByDay[j] < minValue) {
    minValue = usersByDay[j];
    let swap = usersByDay[currentIndex];
    usersByDay[currentIndex] = minValue;
    usersByDay[j] = swap;
    console.log('Меняю местами ' + swap + ' и ' + minValue);
    console.log('Массив сейчас: ' + usersByDay);
  }
}

console.log('На позиции ' + currentIndex + ' находится минимальный элемент ' +
minValue);
*/
```

Второй блок

```
/*
console.log(usersByDay);
currentIndex = 0;
minValue = usersByDay[currentIndex];

for (let j = currentIndex + 1; j <= usersByDay.length - 1; j++) {
  if (usersByDay[j] < minValue) {
    minValue = usersByDay[j];
    let swap = usersByDay[currentIndex];
    usersByDay[currentIndex] = minValue;
    usersByDay[j] = swap;
    console.log('Меняю местами ' + swap + ' и ' + minValue);
    console.log('Массив сейчас: ' + usersByDay);
  }
}

console.log('На позиции ' + currentIndex + ' находится минимальный элемент ' +
minValue);
*/
```

Завершаем сортировку

Программа для сортировки массива из четырёх элементов готова. Да, она негибкая, и её придётся переписывать для массивов с другим количеством элементов. Но это нестрашно, ведь вы уже умеете делать рефакторинг.
В текущей реализации много повторяющегося кода, в котором меняется только значение переменной `currentIndex`. Как сделать структуру программы лучше?
Можно взять кусок кода с циклом, который ищет и подставляет на указанное место минимальное значение, и обернуть его в другой цикл. И в этом верхнем, «родительском», цикле наращивать переменную `currentIndex`.
Да. Циклы можно вкладывать друг в друга.
Учтите, что `currentIndex` должна изменяться не от нуля до длины массива, а от нуля до длины массива, уменьшенной на единицу. Вспомните прошлое задание, в котором мы завершили сортировку для массива, дойдя до предпоследнего элемента (ведь последний элемент автоматически оказался максимальным).
Условие выхода из верхнего цикла давайте писать по аналогии с условием выхода из вложенного цикла:

```
// Цикл до предпоследнего элемента
currentIndex <= usersByDay.length - 2

// Цикл до последнего элемента
j <= usersByDay.length - 1
```

Задание

1. Удалите весь код для сортировки со второго и третьего элементов.
2. Затем весь код после второй строчки оберните в цикл, который увеличивает переменную `currentIndex` с нуля до `usersByDay.length - 2` включительно. Значение `currentIndex` должно увеличиваться на единицу после каждой итерации.
3. Внутри этого цикла удалите дублирующееся объявление переменной `currentIndex`

JS на момент задания

```
let usersByDay = [4, 2, 1, 3];
console.log(usersByDay);

let currentIndex = 0;
let minValue = usersByDay[currentIndex];

for (let j = currentIndex + 1; j <= usersByDay.length - 1; j++) {
  if (usersByDay[j] < minValue) {
    minValue = usersByDay[j];
    let swap = usersByDay[currentIndex];
    usersByDay[currentIndex] = minValue;
```

```
    usersByDay[j] = swap;
    console.log('Меняю местами ' + swap + ' и ' + minValue);
    console.log('Массив сейчас: ' + usersByDay);
  }
}

console.log('На позиции ' + currentIndex + ' находится минимальный
элемент ' + minValue);
// Завершите цикл здесь


// Сортировка со второго элемента

console.log(usersByDay);
currentIndex = 1;
minValue = usersByDay[currentIndex];

for (let j = currentIndex + 1; j <= usersByDay.length - 1; j++) {
  if (usersByDay[j] < minValue) {
    minValue = usersByDay[j];
    let swap = usersByDay[currentIndex];
    usersByDay[currentIndex] = minValue;
    usersByDay[j] = swap;
    console.log('Меняю местами ' + swap + ' и ' + minValue);
    console.log('Массив сейчас: ' + usersByDay);
  }
}

console.log('На позиции ' + currentIndex + ' находится минимальный
элемент ' + minValue);


// Сортировка с третьего элемента

console.log(usersByDay);
currentIndex = 2;
minValue = usersByDay[currentIndex];

for (let j = currentIndex + 1; j <= usersByDay.length - 1; j++) {
  if (usersByDay[j] < minValue) {
    minValue = usersByDay[j];
    let swap = usersByDay[currentIndex];
    usersByDay[currentIndex] = minValue;
    usersByDay[j] = swap;
    console.log('Меняю местами ' + swap + ' и ' + minValue);
    console.log('Массив сейчас: ' + usersByDay);
  }
}
```

```
console.log('На позиции ' + currentIndex + ' находится минимальный
элемент ' + minValue);
```

Тестируем сортировку

Вот теперь программа сортировки точно завершена. Осталось её как следует протестировать.
Лучше всего тестировать программу в обычных и в так называемых «граничных» условиях.
Программе сортировки очень «непривычно» получать на вход уже отсортированные по возрастанию или убыванию массивы. Поэтому эти условия для неё граничные. А обычными, даже обыденными, условия будут, если на входе окажутся неотсортированные массивы разной длины.
В программе в качестве переменных циклов теперь используются `i` и `j`. Это традиционная «не-разлей-вода» парочка имён переменных, которая часто используется во вложенных циклах

Задание

1. Приведем скрипт в первичное состояние.
2. После цикла выведите в консоль массив `usersByDay`.
3. Измените исходное значение `usersByDay` на `[5, 4, 3, 2, 1, 0]`.
4. Измените исходное значение `usersByDay` на `[0, 1, 2, 3, 4, 5]`.
5. Измените исходное значение `usersByDay` на `[812, 1360, 657, 1247]`.
6. Измените исходное значение `usersByDay` на `[812, 1360, 657, 1247, 165]`.

JS на момент задания

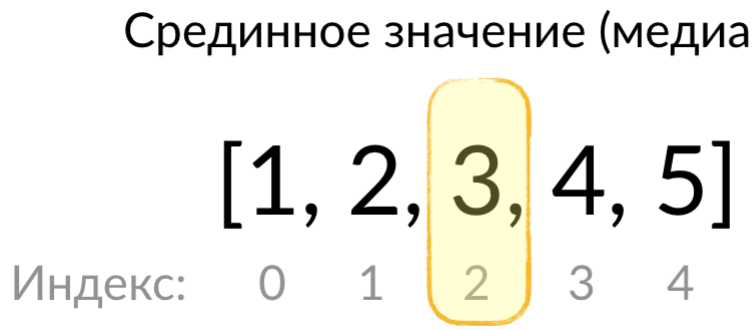
```
let usersByDay = [4, 2, 1, 3];
console.log(usersByDay);

for (let i = 0; i <= usersByDay.length - 2; i++) {
  let minValue = usersByDay[i];

  for (let j = i + 1; j <= usersByDay.length - 1; j++) {
    if (usersByDay[j] < minValue) {
      minValue = usersByDay[j];
      let swap = usersByDay[i];
      usersByDay[i] = minValue;
      usersByDay[j] = swap;
    }
  }
}
```

Медиана на нечётном количестве элементов

Программа сортировки готова и оттестирована. Теперь переходим к подсчёту медианы. Медиана — это срединное значение, то есть это буквально элемент, расположенный посередине массива.



Медиана отличается от среднего значения устойчивостью к отклонениям:
// Среднее: 3
[1, 2, 3, 4, 5]
// Медиана: 3
[1, 2, 3, 4, 5]

// Среднее: 12
[1, 2, 3, 4, 50]
// Медиана: 3
[1, 2, 3, 4, 50]
Медиану легко получить у массива с нечётным количеством элементов. Для этого нужно правильно посчитать индекс среднего элемента. Попробуем вывести формулу:
// Длина массива 3, индекс среднего элемента 1
[1, 2, 3]

// Длина массива 5, индекс среднего элемента 2
[1, 2, 3, 4, 5]

// Длина массива 7, индекс среднего элемента 3
[1, 2, 3, 4, 5, 6, 7]
Вычитаем из длины массива единицу и делим на два, и индекс найден.
Вы можете использовать массивы данных разной длины, поэтому придётся проверять количество элементов на чётность и в зависимости от этого находить медиану. Для проверки на чётность используем уже знакомый оператор %

Задание

1. В конец программы добавьте проверку на нечётное количество элементов в массиве: `usersByDay.length % 2 !== 0`
2. Если условие выполняется, то добавьте переменную `medianIndex` со значением `(usersByDay.length - 1) / 2`
3. Выведите эту переменную в консоль.

4. В той же ветке присвойте переменной `median` значение `usersByDay[medianIndex]`
5. И тоже выведите её в консоль.

JS на момент задания

```
let usersByDay = [1, 2, 3, 4, 5];
console.log(usersByDay);
let median;
```

Медиана на чётном количестве элементов

- Молодец! Возьми с полки пирожок.
- Но там их два.
- Возьми тот, который посередине.

Примерно та же ситуация происходит с подсчётом медианы, когда в массиве хранится чётное количество элементов. Но в отличие от пирожков, для медианы решение нашли. Если количество элементов чётное, то медиана считается как среднее значение от двух элементов: левого и правого от середины.
// Медиана: 3
[0, 1, 2, 4, 50, 100]

Снова выводим формулы индексов двух элементов: левого и правого от середины.
// Длина 4, индекс левого 1, правого 2
[1, 2, 3, 4]

// Длина 6, индекс левого 2, правого 3
[1, 2, 3, 4, 5, 6]

// Длина 8, индекс левого 3, правого 4
[1, 2, 3, 4, 5, 6, 7, 8]

Делим длину массива на два и вычитаем единицу — левый индекс найден. Делим длину массива на два — правый индекс найден.
Расчёт медианы для чётного количества элементов в массиве добавляем в альтернативной ветке условия.
Этот алгоритм поиска медианы не будет работать на пустых массивах и результатом будет NaN. Всё потому, что в расчёты закрадывается значение undefined, а оно в любых математических операциях даёт NaN

Задание

1. В альтернативной ветке условия добавьте переменную `leftIndex` со значением `usersByDay.length / 2 - 1`
2. И переменную `rightIndex` со значением `usersByDay.length / 2`
3. Выведите обе переменные в консоль.
4. В той же ветке условия присвойте переменной `median` значение `(usersByDay[leftIndex] + usersByDay[rightIndex]) / 2`
5. И тоже выведите её в консоль.

Сборка

Сейчас нужно дополнить нашу программу двумя блоками, написанными ранее. Во-первых, добавим код для сортировки массива. Во-вторых, добавим код для вычисления медианы.
И останется только вывести значение медианного значения в консоль, чтобы увидеть, как оно отличается от среднего значения.

Задание

1. Раскомментируйте блок с сортировкой массива. Можете также вывести массив в консоль, чтобы убедиться, что он отсортировался правильно.
2. Затем расскомментируйте блок с расчётом медианы.
3. В конце программы выведите в консоль сообщение `'Медианная посещаемость: ' + median`

JS на момент задания

```
let expectedUsers = 1000;

let usersByDay = [817, 581, 1370, 752, 1247, 681, 1120, 915, 875,
1341, 757, 610, 812, 741, 1139, 812, 638, 877, 1242, 1159, 1372,
1170, 845, 1289, 515, 1247, 769, 1261, 2805, 1201];

// Суммируем посещаемость
let totalUsers = 0;
for (let i = 0; i <= usersByDay.length - 1; i++) {
  totalUsers += usersByDay[i];
}

// Рассчитываем среднее значение посещаемости
let averageUsers = totalUsers / usersByDay.length;
console.log('Средняя посещаемость: ' + averageUsers);

if (averageUsers > expectedUsers) {
  console.log('Посещаемость великолепна. Продолжай в том же
духе!');
} else {
  console.log('Посещаемость так себе. Нужно поднапрячься!');
}

// Сортируем массив
/*
for (let i = 0; i <= usersByDay.length - 2; i++) {
  let minValue = usersByDay[i];

  for (let j = i + 1; j <= usersByDay.length - 1; j++) {
    if (usersByDay[j] < minValue) {
```

```
      minValue = usersByDay[j];
      let swap = usersByDay[i];
      usersByDay[i] = minValue;
      usersByDay[j] = swap;
    }
  }
}
*/

// Рассчитываем медиану
/*
let median;
if (usersByDay.length % 2 !== 0) {
  let medianIndex = (usersByDay.length - 1) / 2;
  median = usersByDay[medianIndex];
} else {
  let leftIndex = usersByDay.length / 2 - 1;
  let rightIndex = usersByDay.length / 2;
  median = (usersByDay[leftIndex] + usersByDay[rightIndex]) / 2;
}
*/
```

Снова выводы!

Отлично, уже сейчас видно, что что-то не так. Ведь средняя посещаемость составляет 1032, а медианная 896 человек в день. А это уже ниже ожидаемого значения в тысячу человек.
Есть одна проблема - чтобы убедиться в подтасовках, надо доказать, что среднее и медиана отличаются сильно, не менее, чем на 10 процентов.
В нашем случае очень подозрительно, когда медиана ниже среднего значения.
Осталось понять, как посчитать проценты.
Для этого нужно поделить значение медианы на среднее значение. Например, если медиана составляет 80, а среднее значение 100, то:
// Медиана составляет 80% от среднего
80 / 100 = 0.8
Переформулируем задачу: если медиана составляет меньше, чем 0.9 от среднего, то есть подозрения в подтасовках.
Осталось дописать проверку и вывести рекомендации в консоль

Задание

1. В конец программы добавьте проверку, что `median / averageUsers < 0.9`
2. Если условие выполняется, выведите в консоль сообщение `'Есть подозрения в подтасовках!'`
3. Иначе выведите в консоль сообщение `'Подозрений в подтасовках нет!'`

Конспект «Массивы». Раздел 2

Сортировка массива

```
let numbers = [12, 3, 7, 9, 10, 5];
for (let i = 0; i <= numbers.length - 2; i++) {
  let minValue = numbers[i];
  for (let j = i + 1; j <= numbers.length - 1; j++) {
    if (numbers[j] < minValue) {
      minValue = numbers[j];
      let swap = numbers[i];
      numbers[i] = minValue;
      numbers[j] = swap;
    }
  }
}
console.log(numbers); // Выведет: [3, 5, 7, 9, 10, 12];
```

Массив с числами numbers сортируется по возрастанию элементов. На каждой итерации мы сравниваем minValue с остальными элементами массива. Если какой-то из них окажется меньше, чем minValue, он запишется в minValue, перезаписав старое значение, и переместится в начало массива. Переменная swap — вспомогательная переменная, с помощью которой мы можем поменять элементы местами.

Поиск медианы массива

```
let median;
if (usersByDay.length % 2 !== 0) {
  let medianIndex = (usersByDay.length - 1) / 2;
  median = usersByDay[medianIndex];
} else {
  let leftIndex = usersByDay.length / 2 - 1;
  let rightIndex = usersByDay.length / 2;
  median = (usersByDay[leftIndex] + usersByDay[rightIndex]) / 2;
}

console.log(median);
```

Физкультура. Прыжки в длину

Итак, отработки пропущенной физкультуры мы уже умеем рассчитывать. Также, благодаря формулам ИМТ мы научились правильно питаться и вести учет необходимых данных. Давайте попробуем усилить наши тренировки и заявимся на соревнования по прыжкам в длину.
Отбор на соревнования жёсткий, квалифицироваться на них тяжело, поэтому нам нужна программа, которая проводит квалификацию по правилам чемпионата.
Впрочем, вот техническое задание:
В течение тренировки вы делаете несколько прыжков и собираете длины прыжков в массив `attempts`.
Квалификационное значение хранится в переменной `qualificationDistance`.
Программа должна выбрать три лучших прыжка, а затем посчитать среднее значение этих трёх прыжков и записать его в переменную `averageBest`.
Если среднее от лучших трёх прыжков больше квалификационного значения, то квалификацию пройдена и переменная `qualified` должна содержать `true`. Если квалификация не пройдена, то в `qualified` должно быть `false`.

Три лучших значения вы можете находить по-разному. Можно отсортировать массив по возрастанию и взять три последних элемента.
Можете устроить себе испытание и попробовать отсортировать массив по убыванию. Тогда после сортировки лучшие три значения будут находиться в начале массива.

JS стартовый

```
let qualificationDistance = 200;
let attempts = [120, 150, 160, 201, 203, 180, 202];
let qualified = false;
let averageBest = 0;
```

Тесты

Первый тест. Прыжки:
[120,150,160,201,203,180,202]
Ожидаем, что квалификация пройдена: значение qualified равное true, значение averageBest равное 202 совпадает с ожидаемым значением.

Второй тест. Прыжки:
[201,199,191,150]
Ожидаем, что квалификация не пройдена: значение qualified равное false, значение averageBest равное 197 совпадает с ожидаемым значением.

Третий тест. Прыжки:
[220,220,220]
Ожидаем, что квалификация пройдена: значение qualified равное true, значение averageBest равное 220 совпадает с ожидаемым значением.

Четвёртый тест. Прыжки:
[202,110,203,150,162,120]
Ожидаем, что квалификация не пройдена: значение qualified равное false, значение averageBest равное 189 совпадает с ожидаемым значением.