**Ключевое слово *each* в языке М Power Query**

Это сокращенный перевод статьи [Маркуса Краучера](https://www.linkedin.com/in/marcus-croucher/?originalSubdomain=nz), опубликованной в блоге [excelguru.ca](https://www.excelguru.ca/blog/2018/01/09/each-keyword-power-query/). Изложение ведется от первого лица (Маркуса).

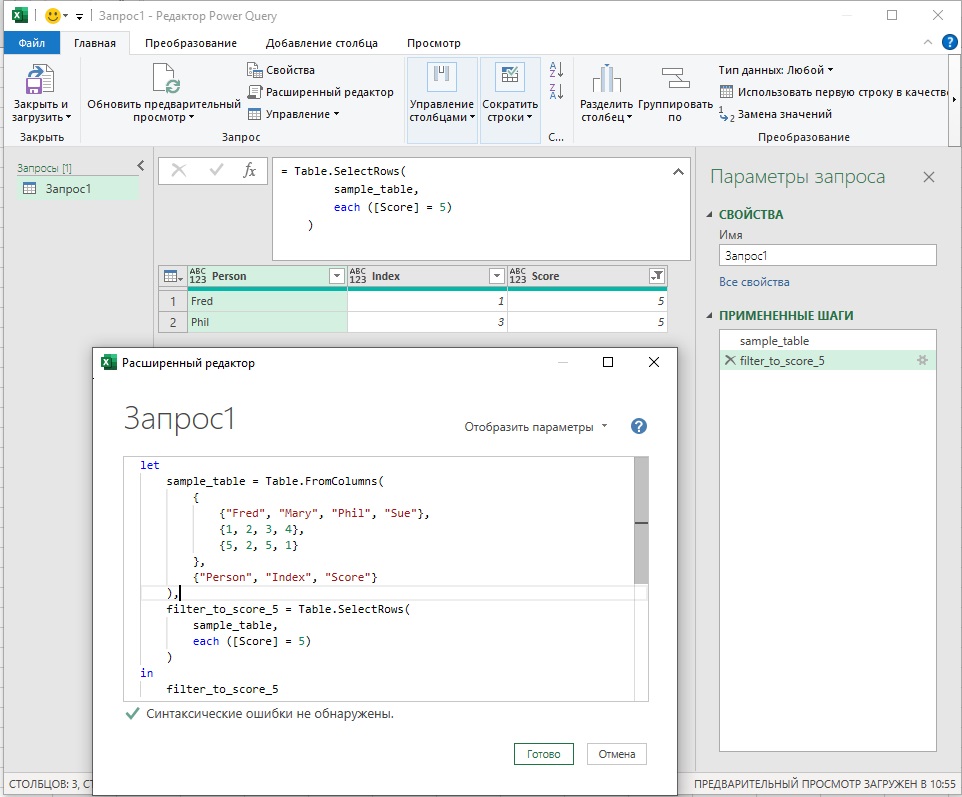


Рис. 1. Фильтрация таблицы на основе *each*

#### Предположения

В этой статье я предполагаю, что у вас есть опыт работы с расширенным редактором и вы понимаете [базовые структуры данных](https://docs.microsoft.com/ru-ru/powerquery-m/expressions-values-and-let-expression#Anchor_3) и способы доступа к ним: Table, List (columns) – обозначается как {item one, item two, etc.} и доступен по [column header], Record (rows) – обозначается как [category1: data1, category2: data2, etc] и доступен по {row number}

Копируйте примеры кода и вставляйте в пустой запрос, чтобы проверить, как он работает.

#### Когда используется ключевое слово each?

Ключевое слово each в Power Query часто попадает в код при работе с интерфейсом. Например, при фильтрации таблицы на основе значения в столбце (см. рис. 1):

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

filter\_to\_score\_5 = Table.SelectRows(

sample\_table,

each ([Score] = 5)

)

in

filter\_to\_score\_5

Мы хотим отфильтровать строки, содержащие в столбце *Score* значение 5. Но, как это работает? [Score] – это целый столбец. Как мы можем сделать целый столбец равным одному значению?

Слово *each* также появляется в контексте пользовательских столбцов. Создадим пользовательский столбец, объединяющий индекс и имя:

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

add\_ordered\_name\_column = Table.AddColumn(

sample\_table,

"ordered\_name",

each Number.ToText([Index]) & " " & [Person]

)

in

add\_ordered\_name\_column

Изображение выглядит как стол

Автоматически созданное описание

Рис. 2. Пользовательский столбец

#### Итак, что такое ключевое слово each в Power Query?

В [спецификации](https://docs.microsoft.com/ru-ru/powerquery-m/understanding-power-query-m-functions) языка указано:

Ключевое слово *each* используется для удобного создания простых функций. "each ..." — это синтаксический сахар для сигнатуры функции, которая принимает параметр "(\_) => ...".

Ключевое слово *each* удобно использовать совместно с оператором подстановки, который по умолчанию применяется к \_.

Например, запись each [CustomerID], аналогичная записи each \_[CustomerID], которая аналогичная записи (\_) => \_[CustomerID].

… и это мало что проясняет.

Оказывается, нужно понять три вещи, чтобы разобраться с *each*:

1. Функции – еще один класс объектов
2. \_ (нижнее подчеркивание) является временной переменной
3. Анонимные функции (и *each* среди них)

#### Функции – еще один класс объектов

В Excel вы привыкли к тому, что любая функция (например, СУММ) немного вещь в себе. Функции нельзя изменять, и они поставляются полностью готовыми к применению. Можно создавать пользовательские функции с помощью VBA, но они совсем не похожи на встроенные функции Excel.[[1]](#footnote-1) В Power Query функции можно рассматривать просто как еще один *объект* – или как другой тип данных. Это означает, что функции могут быть:

* присвоены переменной и/или переименованы. Точно так же, как мы можем сделать что-то вроде variable = 5 в Power Query, мы можем сделать что-то вроде variable = function;
* легко созданы пользователем;
* использоваться в качестве аргумента другой функции.

Чтобы понять это, нам нужно различать *вызов функции* (использование ее в нашем коде) и *обращение* к ней (чтобы дать ей имя или использовать ее в другой функции). Как и в других языках программирования, для вызова функции мы используем круглые скобки в конце:

*function()*

Если мы хотим сослаться на функцию, мы просто опускаем круглые скобки:

*function*

Вот как можно переименовать функцию. Сначала я беру одну из встроенных функций – [List.Sum](https://docs.microsoft.com/ru-ru/powerquery-m/list-sum), которая в качестве аргумента получает *список* (фактически *столбец*) и вычисляет сумму всех его элементов. Затем я создаю таблицу – sample\_table. И, наконец, вычисляю сумму одного из столбцов таблицы – [Score], используя функцию, определенную в первой строке – sum\_column.

let

sum\_column = List.Sum,

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

sum = sum\_column(sample\_table[Score])

in

sum

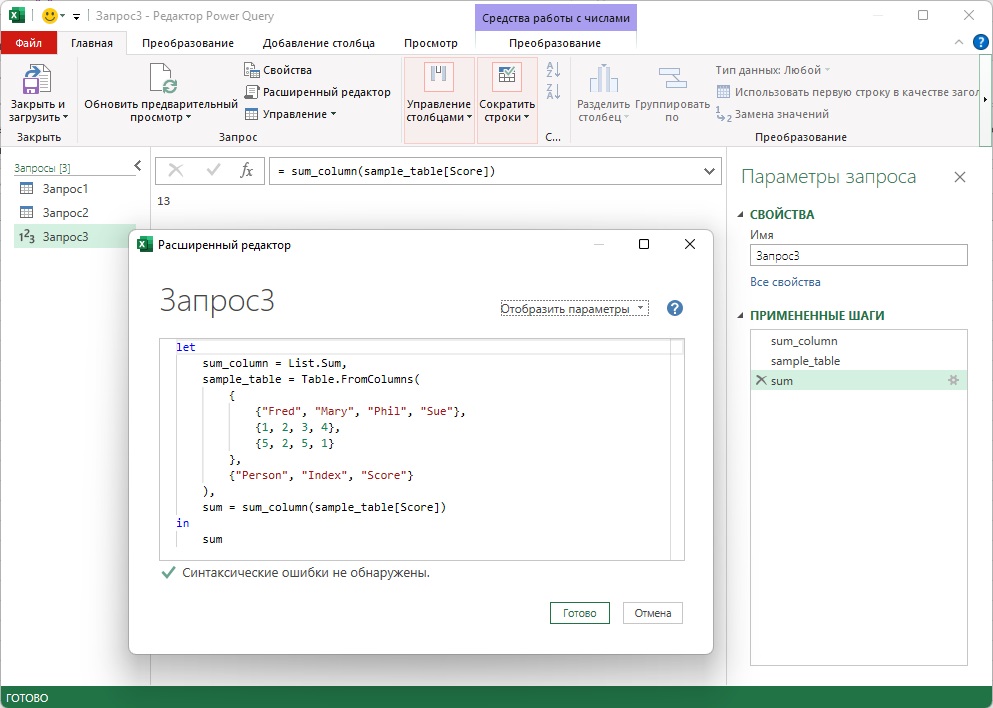


Рис. 3. Переименование функции

#### Как создать функцию

В Power Query синтаксис для создания функции:

(переменная) => тело функции

Тело функции похоже на любой другой запрос, который возвращает значение, но вместо того, чтобы возвратиться в интерфейс Power Query, тело функции возвращает значение тому, кто его вызвал.

Если мы хотим присвоить функции имя, оно должно предшествовать приведенному выше шаблону. Создадим простую функцию, которая принимает число и умножает его на два.

multiply\_by\_two = (number) =>

let

output = number \* 2

in

output

Для использования этого фрагмента включим его в запрос:

let

multiply\_by\_two = (number) =>

let

output = number \* 2

in

output,

x = 5,

y = multiply\_by\_two(x)

in

y

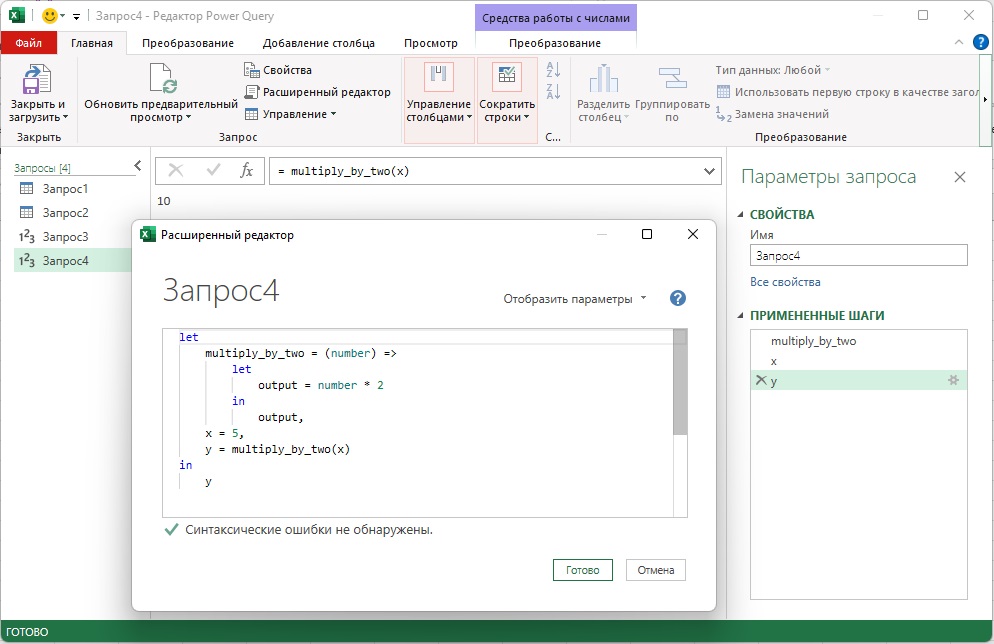


Рис. 4. Запрос, использующий созданную нами функцию

#### Функции в качестве аргумента другой функции

Рассмотрим в качестве примера функцию фильтрации [Table.SelectRows](https://docs.microsoft.com/ru-ru/powerquery-m/table-selectrows), которая принимает другую функцию в качестве входных данных:

Table.SelectRows(table as table, **condition as function**) as table

Функция возвращает таблицу, содержащую только строки, соответствующие условию.

Таким образом, функция Table.SelectRows ожидает в качестве аргументов таблицу и функцию. Как это работает? Table.SelectRows применяет полученную в качестве аргумента функцию к каждой строке таблицы и ожидает от функции вернет *true* или *false*. Затем использует этот ответ, чтобы решить, следует ли сохранить или удалить строку. Напомним, что мы можем получить доступ к элементам строки (записи), указав имя столбца: запись [имя столбца].

Зададим функцию filterer\_score\_two\_plus, которую затем передадим в Table.SelectRows:

let

filterer\_score\_two\_plus = (record) =>

let

value = record[Score],

result = value >= 2

in

result,

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}),

filtered = Table.SelectRows(

sample\_table,

filterer\_score\_two\_plus

)

in

filtered

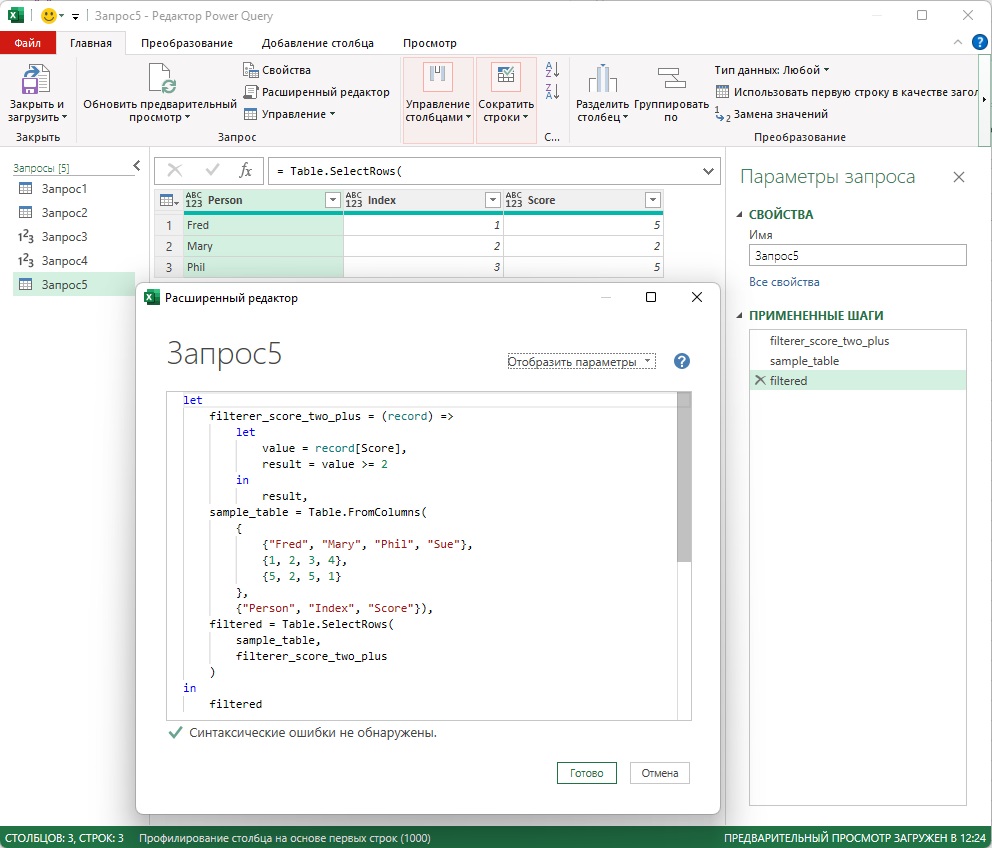


Рис. 5. Фильтрация таблицы с помощью функции filterer\_score\_two\_plus

Сначала я создал функцию filterer\_score\_two\_plus, которая принимает запись, извлекает значение из столбца [Score] и, если оно больше или равно двум, возвращает значение true. Затем я создал таблицу sample\_table. Далее применил фильтр (используя функцию Table.SelectRows) к таблице sample\_table, предоставляя только что созданную функцию filterer\_score\_two\_plus в качестве второго аргумента. Результат – таблица со всеми строками, в которых значение в столбце [Score] больше или равно двум.

Так вот, оказывается, что в языке М есть более быстрый и простой способ создания таких функций. Способ был придуман, поскольку создавать такие «одноразовые» функции приходится довольно часто.

#### \_ в качестве временной переменной

Использование \_ в качестве одноразовой переменной распространено в целом ряде языков программирования, включая Python. Обычно \_ используется для обозначения переменной, которая больше не будут использоваться, и поэтому не стоит утруждать себя придумыванием имени.

Вот запрос, создающий таблицу, которая присваивается переменной с именем \_.

let

\_ = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

)

in

\_

Для Python использование \_ – это просто соглашение. В Power Query расширили эту функциональность. Допустим, нам нужен столбец имен из приведенной выше таблицы. Обычно мы можем сделать это, выбрав столбец по имени – [column\_name].

let

\_ = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

)

in

\_[Person]

Оказывается, мы можем опустить \_ в этой записи. Если мы просто укажем [Person] Power Query поймет, что, таблица, на которую мы ссылаемся, называется \_:

let

\_ = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

)

in

[Person]

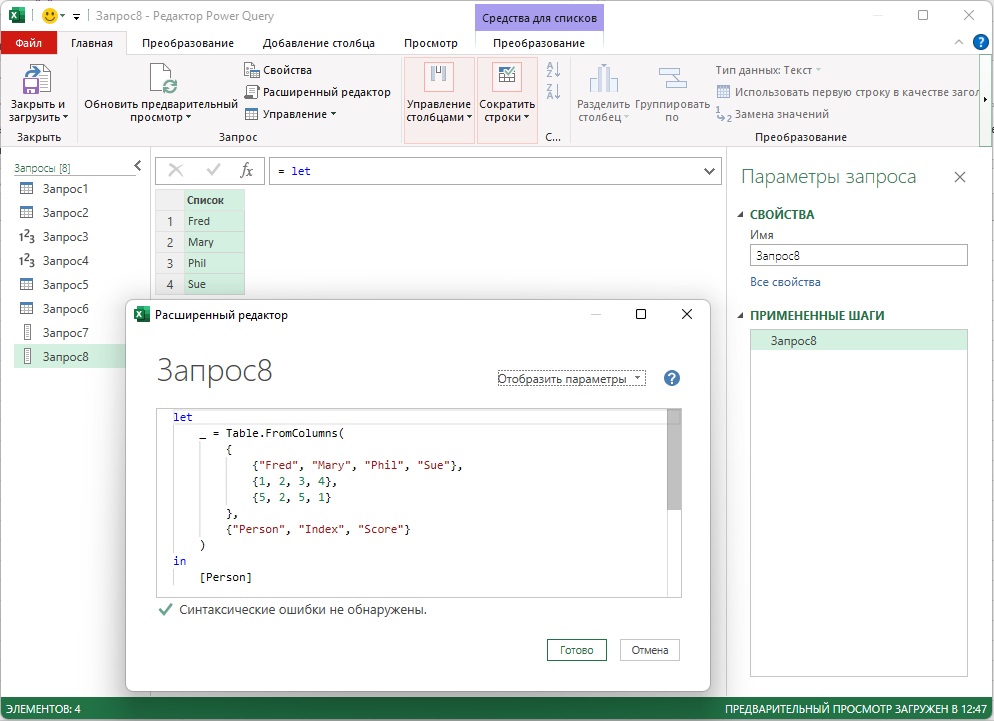


Рис. 6. Если мы опустим переменную \_, Power Query сообразит, что таблица, на которую мы ссылаемся, называется \_

Я бы не рекомендовал это в качестве общей практики, поскольку эта особенность плохо документирована. А явное обычно лучше, чем неявное. Тем не менее, такое поведение PQ обеспечивает красивый код при использовании с ключевым словом *each*. Обратите внимание, что этот метод работает только для выбора столбца [column\_name], но не для выбора строки {row\_number}, поскольку Power Query интерпретирует {row\_number} как новый список.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 7. Здесь знак \_ перед {1} необходим, иначе запрос вернет не то, что вы ожидаете – вторую строку таблицы

#### Применение переменной \_

С учетом этой концепции давайте пересмотрим наш запрос фильтрации (см. рис. 5):

let

filterer\_score\_two\_plus = (\_) =>

let

result = [Score] >= 2

in

result,

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}),

filtered = Table.SelectRows(

sample\_table,

filterer\_score\_two\_plus

)

in

filtered

И, наконец, давайте поместим определение функции filterer\_score\_two\_plus прямо в Table.FromColumns и избавимся от let и in (которые требуются только, если в вычислении несколько шагов):

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}),

filtered = Table.SelectRows(

sample\_table,

(\_) => [Score] >= 2

)

in

filtered

Получилось кратко и понятно. Но конструкция (\_) => всё еще пугает, если вы не знакомы с определениями функций, а символ определения функции => похож до смешения с оператором больше или равно >=.

#### Анонимные функции

Мы довольно часто создаем одноразовые функции для использования другими функциям. Кажется глупым использовать весь синтаксис создания и присвоения имени функции, если она больше не будет использоваться. На помощь приходит концепция анонимных функций, которые определены, но не названы. Они используются сразу же после их создания. В Python они известны как лямбда-функции.

Мы можем использовать ключевое слово *each* в Power Query для определения функции. *each* просто сводит к минимуму синтаксис для создания функции, предоставляя имя входной переменной по умолчанию \_ и устраняя необходимость в =>. Вместо…

(\_) =>

let

result = [Score] >= 2

in

result

… можно использовать…

each [Score] >= 2

Вы все равно можете использовать определение функции…

filterer = each [Score] >= 2

…но ключевое слове each в Power Query гораздо полезнее, если мы используем его встроенным в основную функцию. Итак, предыдущий пример примет вид, который уже выглядит знакомо для пользователей:

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}),

filtered = Table.SelectRows(

sample\_table,

each [Score] >= 2

)

in

filtered

#### Итак, что же мы обнаружили?

Мы можем использовать ссылку на [имя столбца] вместо ссылки на имя таблицы и имя столбца \_ [имя столбца]. Конструкция *each* проходит по каждой записи таблицы, ожидая ответа true / false. К сожалению, всё это не очень хорошо задокументировано.

Теперь, когда мы понимаем контекст использования *each* (текущая запись/строка), мы можем создавать любопытные функции. Иногда для них требуется индексный столбец. Например, мы можем отфильтровать таблицу на основе второй таблицы или добавить новый столбец, который вычитает текущую строку из предыдущей. Вот код для второй идеи:

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

add\_index = Table.AddIndexColumn(

sample\_table,

"raw\_index"

),

add\_difference\_column = Table.AddColumn(

add\_index,

"difference",

each if [raw\_index] = 0

then [Score]

else [Score] - add\_index[Score]{[raw\_index] - 1}

),

delete\_index = Table.RemoveColumns(

add\_difference\_column,

"raw\_index"

)

in

delete\_index

Обратите внимание, при ссылке на столбец – *else [Score]* – не требуется указание на таблицу…

else [Score] - add\_index[Score]{[raw\_index] - 1}

… а при ссылке на строку – add\_index[Score]{[raw\_index] - 1} – требуется. Здесь имя таблицы *add\_index*.

Изображение выглядит как стол

Автоматически созданное описание

Рис. 8. Столбец *difference*, вычитает текущую строку из предыдущей

#### each не работает внутри функции Table.TransformColumns()

К сожалению, внутри Table.TransformColumns() с помощью each нельзя получить доступ к значениям из другого столбца. В соответствии со [спецификацией](https://docs.microsoft.com/ru-ru/powerquery-m/table-transformcolumns) …

Table.TransformColumns(table as table, transformOperations as list, …) as table

… возвращает таблицу из входного table, применяя операцию преобразования к столбцу, указанному в параметре transformOperations с форматом *{имя столбца}, преобразование*.

Давайте поэкспериментируем. Как будет работать код?

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

result = Table.TransformColumns(sample\_table, {}, each \_)

in

result

Изображение выглядит как стол

Автоматически созданное описание

Рис. 9. Table.TransformColumns предоставляет вспомогательной функции значения отдельных ячеек

Похоже, что нашей вспомогательной функции передается только содержимое одной ячейки, а не вся строка. Функция Table.TransformColumns(sample\_table, {}, each \_) берет sample\_table, и применяет вспомогательную функцию each \_ ко всей таблице (третий аргумент является функцией, и поскольку мы предоставили пустой список во втором аргументе, наша вспомогательная функция применяется ко всей таблице).

Таблица осталась прежней. Функции each передается содержимое одной ячейки, нет никакого внешнего контекста. К сожалению, это означает, что у нас нет возможности ссылаться на внешние столбцы.

Внутри Table.AddColumn вспомогательная функция each получает всю строку (запись). Протестируйте код:

let

sample\_table = Table.FromColumns(

{

{"Fred", "Mary", "Phil", "Sue"},

{1, 2, 3, 4},

{5, 2, 5, 1}

},

{"Person", "Index", "Score"}

),

result = Table.AddColumn(sample\_table, "new\_column", each \_)

in

result

В новом столбце появились записи:

Изображение выглядит как стол

Автоматически созданное описание

Рис. 10. Table.AddColumn предоставляет вспомогательной функции целые записи (строки)

Мы можем использовать эти записи в столбце *new\_column* для создания контекста, например, создав столбец индекса.

Похоже, если мы хотим объединить информацию из нескольких столбцов, мы можем это сделать только добавив новый столбец, но не изменяя столбец на месте.

1. С недавних пор можно создавать пользовательские функции прямо в Excel с использованием встроенной функции [LAMBDA](https://baguzin.ru/wp/?p=24759). [↑](#footnote-ref-1)