Apuntes de Lenguaje M

1. Introducción

Power Query es una herramienta de transformación y extracción de datos muy poderosa que viene integrada en Excel 2016 (o posterior), Excel para Office 365 y Power BI.

Se puede encontrar en la pestaña Datos en la sección Obtener y transformar datos.

2. ¿Qué es el código M?

La M significa Data Mash -up, ya que Power Query se trata de conectarse a varias fuentes de datos diferentes y "Mezclarlas".

El código M es el lenguaje de Power Query. Cuando crea una transformación de datos en la interfaz de usuario del editor de consultas de Power, Excel escribe el código M correspondiente para la consulta.

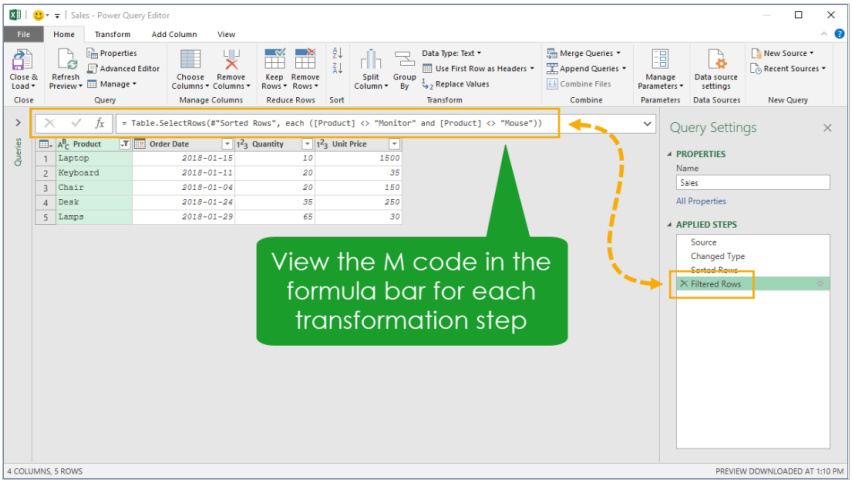
M es un lenguaje funcional, lo que significa que está escrito principalmente con funciones que se llaman para evaluar y devolver resultados.

El código M viene con una biblioteca muy grande de funciones predefinidas disponibles y también puede crear las suyas propias.

3. ¿Dónde se puede escribir el código M de Power Query?

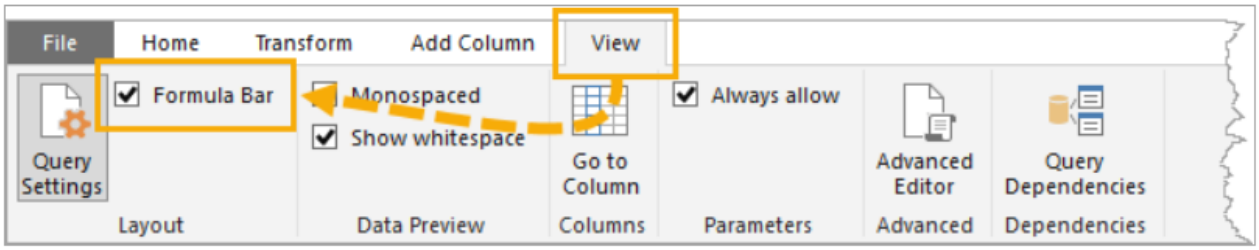
Si desea comenzar a escribir o editar código M, necesitará saber dónde puede hacerlo. Hay dos lugares donde es posible, en la barra de fórmulas o en el editor avanzado.

La barra de fórmulas

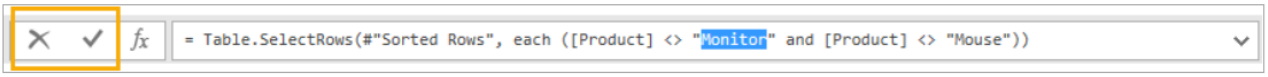


Para cada paso que se crea en la interfaz de usuario del editor se genera un código M correspondiente.

Si no ve la barra de fórmulas, vaya a la pestaña Ver y asegúrese de que la opción Barra de fórmulas esté marcada.



Puede editar el código M para cualquier paso de una consulta haciendo clic en la fórmula y editando el código.

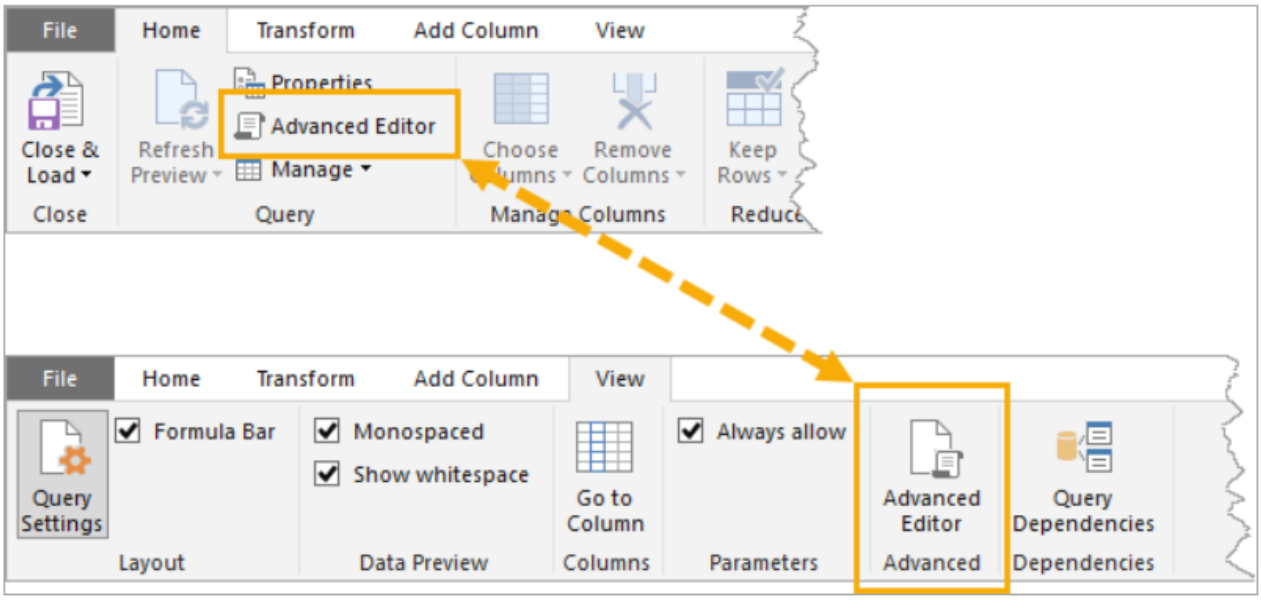


Cuando termine, puede aceptar cualquier cambio haciendo clic en la marca de verificación o presionando **Enter**. También puede descartar sus cambios haciendo clic en la **X** o presionando **Esc**.

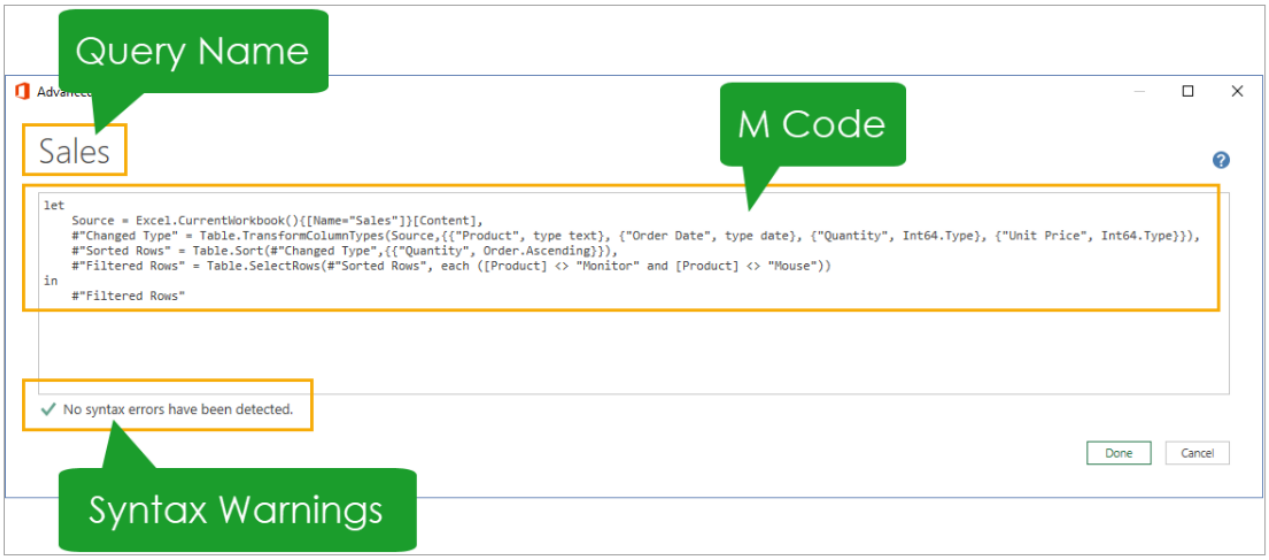
También puede crear pasos completamente nuevos en su consulta con la barra de fórmulas haciendo clic en el símbolo **fx** al lado de la barra de fórmulas. Esto creará un nuevo paso que hace referencia al paso anterior y a continuación puede crear cualquier código M que necesite.

El editor avanzado

En la barra de fórmulas solo se muestra el código M del paso seleccionado actual, pero si se desea ver y editar el código M se realiza a través del editor avanzado.



Puede abrir el editor avanzado desde dos lugares; desde la pestaña Inicio o la pestaña Ver, presione el botón Editor avanzado.



A pesar del apodo "avanzado", el editor es el editor de código más básico que verá y no contiene (todavía) ninguna función de autocompletado, resaltado de sintaxis o formato automático de IntelliSense.

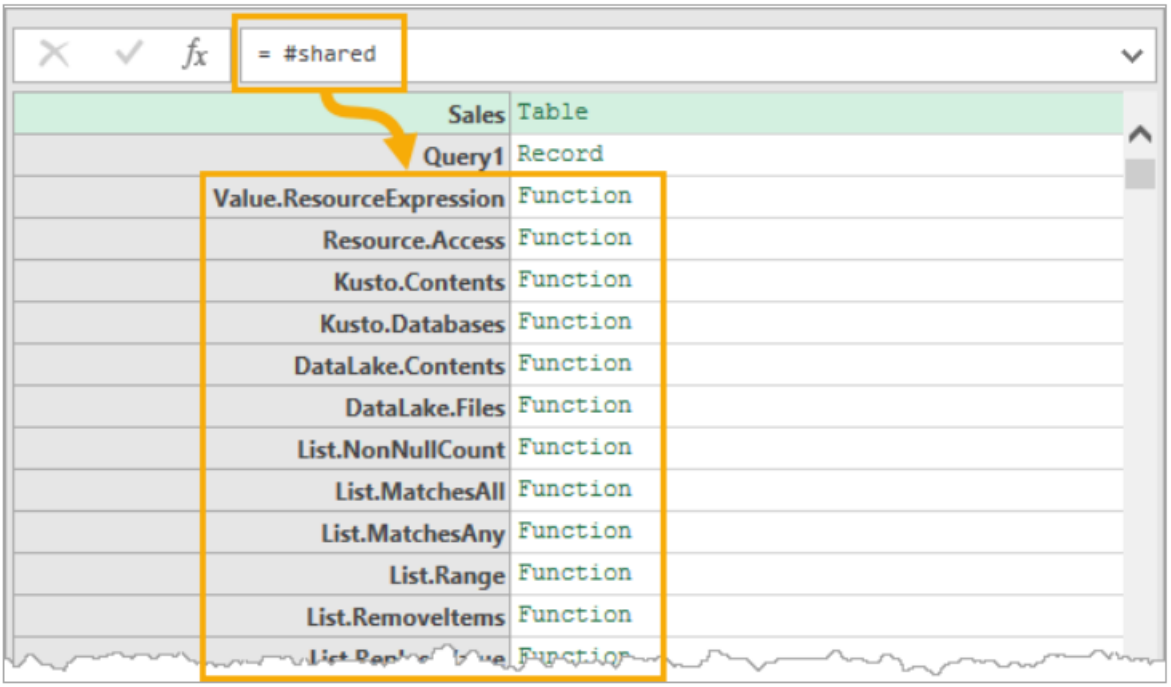
El editor avanzado mostrará el nombre de la consulta, el código M para la consulta y una advertencia sobre cualquier violación de sintaxis en el código M.

4. Biblioteca de funciones estándar

Dado que el código M es un lenguaje funcional, el código M viene con una gran biblioteca de funciones predefinidas llamada **biblioteca estándar.**

Puede encontrar información sobre todas las funciones de biblioteca estándar disponibles en la página web de referencia de Power Query M de Microsoft (https://learn.microsoft.com/en-us/powerquery-m/), incluida la sintaxis de funciones y ejemplos.

La biblioteca estándar también se puede explorar desde el editor de Power Query usando la palabra clave #shared.



Podrá explorar todas las funciones disponibles haciendo clic en la palabra Funtion a la derecha del nombre de la función. Encontrará la misma sintaxis y ejemplos que la página web de referencia.

5. Sensibilidad de mayúsculas y minúsculas

Una de las primeras cosas que se debe tener en cuenta al escribir código M es que es un lenguaje que distingue entre mayúsculas y minúsculas.

Esto significa que **x** no es lo mismo que **X** ni **abc** es lo mismo que **ABC**. Esto es así para cualquier valor, variable, función, etc.

6. Expresiones y valores en Power Query

Power query contiene Expresiones y Valores.

Una **expresión** es algo que se puede evaluar para devolver un valor:

✔ 1 + 1 es una expresión que devuelve el valor 2

Un **valor** es un contenedor de dato.

Los valores pueden ser valores:

- únicos
- números
- texto
- lógicos
- nulos
- binarios
- de fecha
- hora
- fecha hora
- fecha hora zona
- duraciones

Los valores también pueden tener estructuras más complejas:

- listas
- registros
- tablas

Además, pueden existir valores que sean una combinación de listas, registros y tablas:

- listas de listas
- tablas de listas
- tablas de tablas, etc.

Todas estas son estructuras de valores posibles.

Valores literales únicos

Los valores literales únicos son el componente básico de todos los demás valores.

- 123.45 es un valor numérico.
- "Hello World!" es un valor de texto.
- True es un valor lógico.
- null representan la ausencia de un valor.

Valores intrínsecos únicos

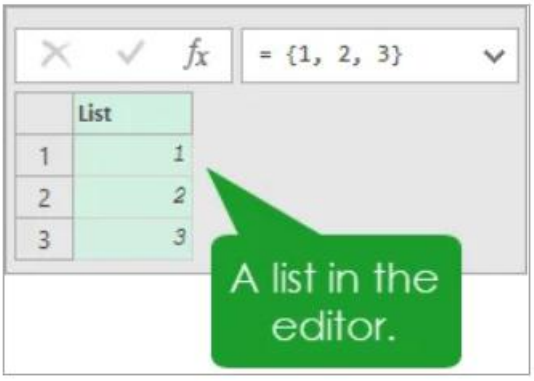
Los valores intrínsecos se construyen utilizando las diversas funciones intrínsecas.

- #time(hours, minutes, seconds)
- #date(years, months, days)
- #datetime(years, months, days, hours, minutes, seconds)
- #datetimezone(years, months, days, hours, minutes, seconds, offset-hours, offset-minutes)
- #duration(days, hours, minutes, seconds)

Por ejemplo, para construir la fecha 2018-12-31, necesitaría construirla usando la función intrínseca: #date(2018, 12, 31)

Valores estructurados

1. Listas

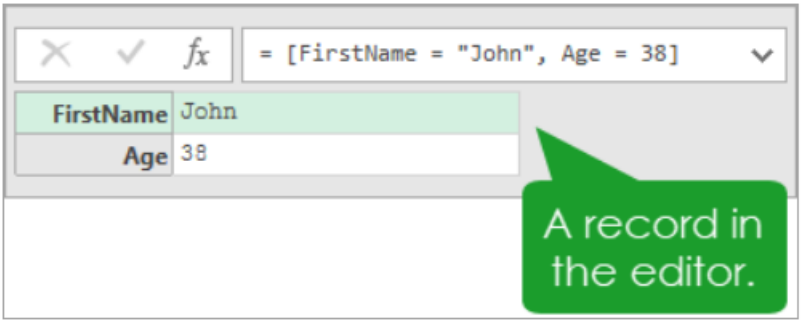


Una lista es una secuencia ordenada de valores con las siguientes características:

- Una lista se define de la siguiente manera: {1, 2, 3} Dado que el orden es importante, esta lista no es la misma que {3, 2, 1}.
- {"Hello", "World"} es una lista que contiene el texto "Hello"y "World".

- Las listas de listas también son posibles: {{1, 2}, {3, 4, 5}} La primera lista contiene el número 1y 2 y la segunda lista contiene los números 3, 4 y 5.
- Puede crear listas secuenciales utilizando el formato {x..y}. {2..5} creará la lista {2, 3, 4, 5}.
- Esto también funciona para caracteres de texto. {"a".."d"}creará la lista {"a", "b", "c", "d"}.
- También puede tener una lista sin elementos, {} es la lista vacía.
- Dado que las listas están ordenadas, puede hacer referencia a elementos de la lista con un número de índice comenzando desde cero (como los arrays). {1, 2, 3}

2. Registros

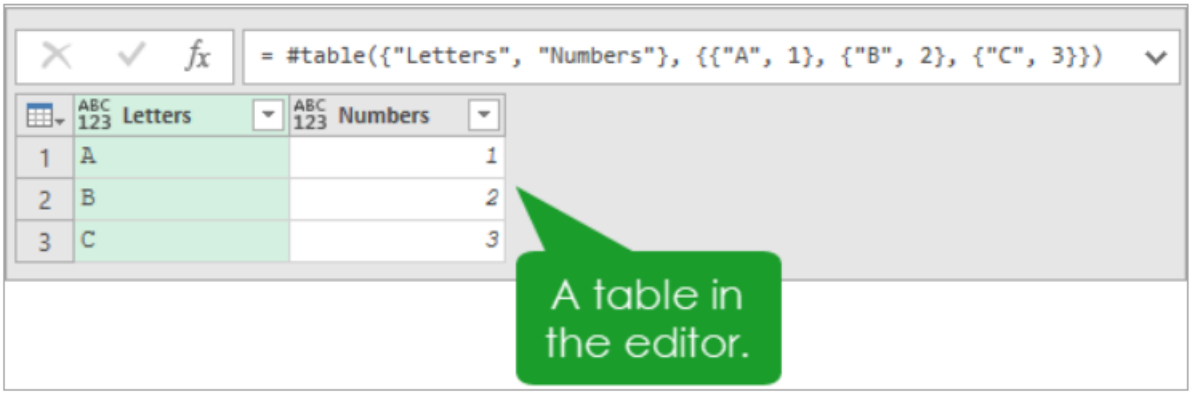


Un Registro es una secuencia ordenada de Campos con las siguientes características:

- Cada campo consta de un **nombre** de campo que identifica de forma exclusiva el campo y un **valor** de campo que puede ser cualquier tipo de valor.
- Puede definir un registro usando corchetes: [FirstName = "John", Age = 38] esto sería un registro con dos campos. El primer campo del registro tiene como nombre FirstName y el valor John. El segundo campo del registro tiene como nombre Age y valor 38.
- También son posibles registros de registros: [Person = [FirstName = "John", Age = 38]] es un registro con un campo de nombre Person y un valor que es un registro.
- Los registros vacíos también son posibles, [] es el registro vacío.

3. Tablas

Una tabla es una secuencia ordenada de filas donde **cada fila es una lista**.



- Las tablas solo se pueden construir utilizando una función intrínseca. Se puede construir una tabla utilizando la función #table() a partir de una lista de columna y una lista de filas.

  #table({"Letters", "Numbers"}, {{"A", 1}, {"B", 2}, {"C", 3}})

  Creará una tabla con 2 columnas, 3 filas y las columnas se llamarán Letters y Numbers.
- Es posible crear una tabla vacía usando listas vacías: #table({}, {})
- Se puede hacer referencia a cualquier valor de una tabla con el índice de la fila y el nombre de la columna.

  #table({"Letters", "Numbers"}, {{"A", 1}, {"B", 2}, {"C", 3}})

  p.e {2}[Letters] devolverá C

7. Expresiones

- Las expresiones son cualquier cosa que pueda evaluar un valor.
- La expresión 1 + 1 se evalúa como 2.
- La expresión 3 > 2 se evalúa como true.
- La expresión "Hello " & "World" se evalúa como "Hello World".
- La expresión Text.Upper("Hello World") se evalúa como "HELLO WORLD".

8. Operadores

Aritmética

+, -, * y / : Estos te permitirán sumar, restar, multiplicar y dividir valores respectivamente. Estos se pueden usar con varios otros tipos de valores que no sean solo números. Por ejemplo, puede agregar una duración a una fecha.

#date(2018,12,25) + #duration(7, 0, 0, 0)se evaluará a 2019-01-01 .

Comparación

<, >, <=, >=, =, <>: Estos permiten comparar valores

- x < y se evaluará como verdadero si x es menor que y.
- x > y se evaluará como verdadero si x es mayor que y.
- x <= y se evaluará como verdadero si x es menor o igual que y.
- x >= y se evaluará como verdadero si x es mayor o igual que y.
- x = y se evaluará como verdadero si x es igual a y.
- x <> ys e evaluará como verdadero si x no es igual a y.

Estos se pueden utilizar con varios tipos de valores. Por ejemplo, puede comparar dos listas con el operador igual.

{1,2,3,4} = {1,2,3} se evaluará como falso ya que las listas no son las mismas.

Concatenación y Fusión

Puede concatenar texto y fusionar listas, registros y tablas mediante el operador &

Por ejemplo:

- "Hello " & "World" se evaluará como "Hello World".
- {1,2,3} & {3,4,5} se evaluará como {1,2,3,3,4,5}.

Lógico

not, and y or: operaciones expresiones que se evalúan como valores booleanos

- not xs e evaluará como verdadero cuando x es falso.
- x and y se evaluará como verdadero cuando tanto x como y sean verdaderos.
- x or y se evaluará como verdadero cuando x o y sean verdaderos.

9. Comentarios

Como cabría esperar de cualquier lenguaje de programación, es posible agregar comentarios a su código.

Hay dos tipos de comentarios posibles en el código M. Comentarios de una sola línea y comentarios de varias líneas.

Comentarios de una sola línea

M code goes here

M code goes here **//This is a single line comment**

M code goes here

Se puede crear un comentario de una sola línea precediendo el comentario con dos caracteres de barra inclinada //. Cualquier cosa en la misma línea antes de esto se interpretará como código M, cualquier cosa después se interpretará como un comentario.

Comentarios de varias líneas

M code goes here **/*This is a comment**

on multiple lines*/ M code goes here

Se puede crear un comentario de varias líneas colocando el comentario entre los caracteres /*y */. Cualquier cosa fuera de estos se interpretará como código M. Cualquier cosa entre estos se interpretará como un comentario.

10. Let Declaración

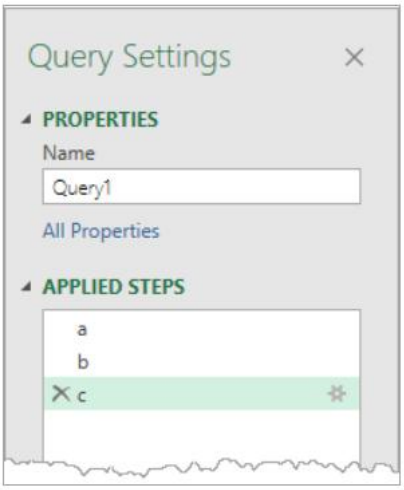
La declaración **let** se programa las líneas de código y en el **in** se ejecutan.

```
let
    a = 1,
    b = 2,
    c = a + b
in
    c
```

```
let
    c = a + b,
    b = 2,
    a = 1
in
    c
```

```
let
    a = 1,
    b = 2
in
    a + b
```

En todas las funciones el resultado será 3. Las expresiones aparecerán como pasos separados en la ventana Pasos aplicados. Cuando se escriben fuera de orden, las expresiones aparecerán como un paso combinado.

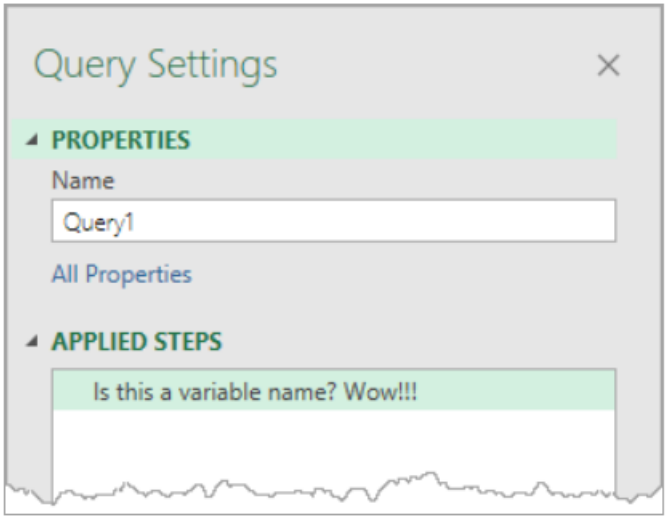


11. Nombres de variables

```
let
    #"Is this a variable name? Wow!!!" = 1 + 1
in
    #"Is this a variable name? Wow!!!"
```

Puede asignar casi cualquier nombre a sus expresiones utilizando los #"" caracteres. Incluso puede utilizar caracteres de espacio y otros caracteres especiales.

La única excepción es el uso de palabras clave reservadas



Los nombres de las variables son los que aparecerán en los Applied steps del editor de consultas, pueden usarse espacios en blanco para los nombres

12. Declaración EACH

Por cada valor del elemento de la columna Number de la tabla:

```
let
    Source = #table({"Numbers"}, {{1}, {2}, {3}, {4}, {5}}),
    #"Added Custom" = Table.AddColumn(Source, "Double", each 2*[Numbers])
in
    #"Added Custom"
```

Es lo mismo que:

```
let
    Source = #table({"Numbers"}, {{1}, {2}, {3}, {4}, {5}}),
    #"Added Custom" = Table.AddColumn(Source, "Double", (_) => 2*_[Numbers])
in
    #"Added Custom"
```

13. Declaraciones If Then Else

El código M es bastante escaso en comparación con otros lenguajes, no existen sentencias como select case o loop disponibles. La única que existe es: if... then... else...

```
if [logical expression to test]
then [do this when true]
else [do this when false]
```

La sintaxis es sencilla y es como la mayoría de los otros lenguajes de programación.

Puede aparecer todo en una línea **(NO HACER)** o puede presentarse en líneas separadas para facilitar la lectura.

14. Declaración Try Otherwise

Pueden ocurrir errores al intentar realizar operaciones que requieren determinados tipos de datos. Por ejemplo, intentar multiplicar un número con un valor de texto dará como resultado un error.

```
let
    Source = #table({"Number", "Number and Text"}, {{2, 2}, {2, "Hello"}}),
    #"Added Custom" = Table.AddColumn(Source, "Product",
                                      each
                                          try
                                              [Number]*[Number and Text]
                                          otherwise
                                          0)
in
    #"Added Custom"
```

Los errores se pueden evitar usando la expresión try... otherwise....

15. Funciones

Una función es una asignación de un conjunto de valores de parámetros a un valor. Junto con las funciones de la biblioteca estándar, el código M le permite crear sus propias funciones.

```
let
    Product = (x,y) => x * y,
    Result = Product(2,3)
in
    Result
```

Esta consulta define una función que multiplica dos números. La consulta llama y evalúa la función con los valores 2 y 3, que se retornará el valor 6.

16. Funciones con parámetros opcionales

En las funciones hay dos tipos de parámetros: un parámetro obligatorio y un parámetro opcional.

✔ **Obligatorios**: siempre deben especificarse

✔ **Opcionales**: no es necesario especificarlos

```
let
    Product = (x, optional y) => if y is null then x else x * y,
    Result = Product(2)
in
    Result
```

En caso de que y no sea nulo la función devolvería un error

17. Funciones recursivas

También es posible escribir una función que se refiera a sí misma utilizando el @operador de ámbito.

```
let
    Fibonacci = (n) =>
        if n = 1  then 1
        else
            if n = 2  then 1
            else @Fibonacci(n-1) + @Fibonacci(n-2),
    Result = Fibonacci(7)
in
    Result
```

* Fibonacci: https://www.youtube.com/watch?v=B6ztvqvZTsk

La secuencia de Fibonacci es un ejemplo de una función que se define recursivamente.

El siguiente número en la secuencia se define como la suma de los dos números anteriores.

Entonces, para obtener el número n, necesita conocer los números (n-1) y (n-2).

Esta función encontrará el n-ésimo número de Fibonacci sumando los (n-1)-ésimo y (n-2)-ésimo número de Fibonacci.

La consulta se evalúa como 13, ya que 13 es el séptimo número de Fibonacci.

18. La funciones Query

Los ejemplos anteriores definen una función dentro de una consulta y se llaman dentro de la misma consulta. También es posible crear funciones para llamarlas desde otras consultas.

```
let FunctionResult = (Argument1, Argument2,...) =>
        let
            /*M code to evaluate in the function goes here*/
        in
            Result
in
    FunctionResult
```

*Tenga en cuenta que necesitará una declaración let... in...dentro de la declaración let... in… de la función

19. BIBLIOGRAFÍA

https://www.howtoexcel.org/m-code/