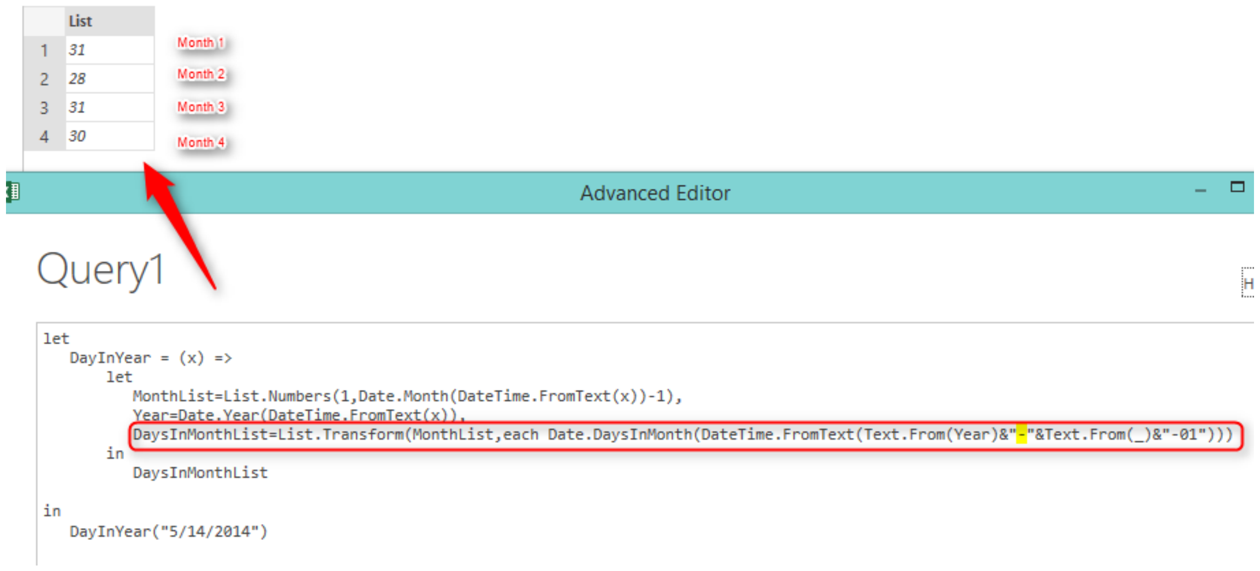
Lab01: Writing Custom Functions in Power Query M



	List
1	31
2	28
3	31
4	30

```
let
   DayInYear = (x) =>
       let
          MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),
          Year=Date.Year(DateTime.FromText(x)),
          DaysInMonthList=List.Transform(MonthList,each Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")))
       in
          DaysInMonthList

in
   DayInYear("5/14/2014")
```

One of the most powerful features of M is that you can write custom functions to re-use part of your code. Custom functions can be single or multiple liner, they will be written in Lambda style syntax. In this post you will learn how to write functions and invoke them in M.

Basic Syntax
Functions in M can be written in this format:

(x) => x+1
This is lambda syntax (that previously used in LINQ if you come from .NET development background). the line above is equal to pseudo-code below:

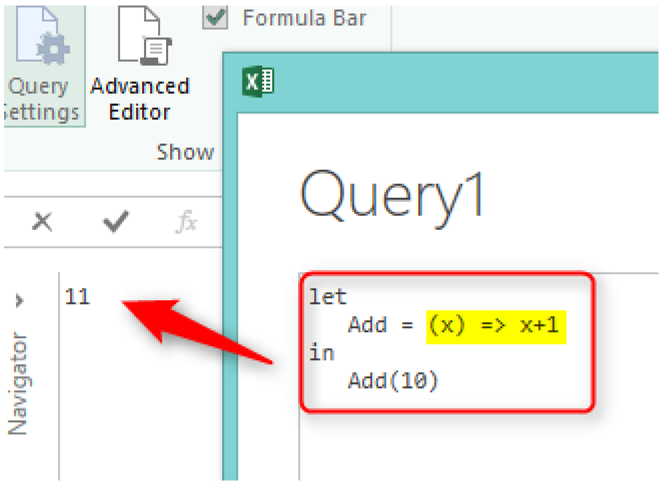
```
Function anonymouss (x)
{
return x+1
}
```

As you see lambda made it much simpler to define the function with just that single line. So, the function above gets a parameter from input and adds 1 to it. Please note that datatype of parameter not defined, so that means if a text be an input, then an error would occur, so you would require to do error handling as well.

So, let’s see how this function works and how we can call this function. script below shows how to define the function and invoke it with a parameter:

```
let
  Add = (x) => x+1
in
  Add(10)
```

Result of above expression is: 11



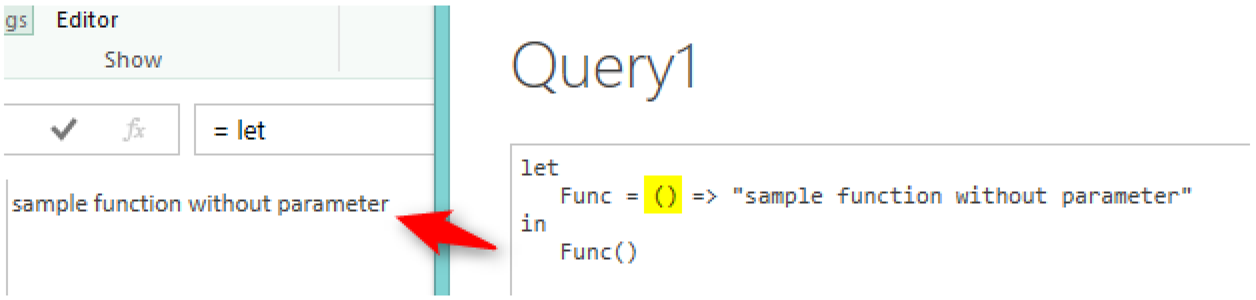
```
let
   Add = (x) => x+1
in
   Add(10)
```

In example above we’ve named the function as "Add" and then we simply call that with Add(inputparam).

Parameters
If you want to define a function with more parameters, then simply add parameters as below:

(x,y,z) => x+y+z
you can also define a function without parameters, as below:

() => "sample function without parameter"
This function gets no parameters and returns the text "sample function without parameter".

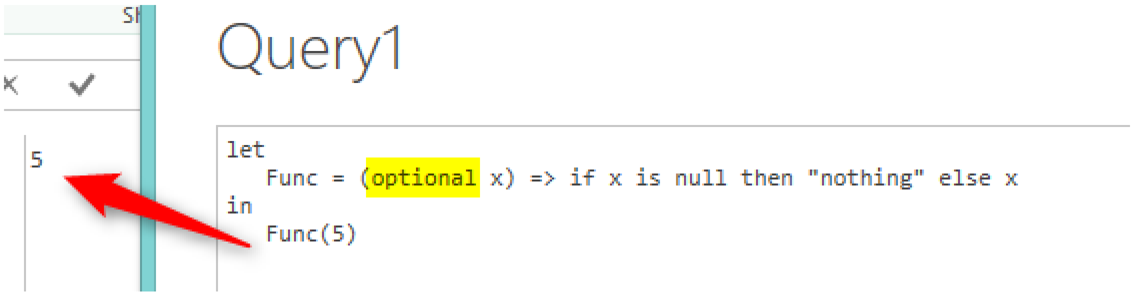


```
sample function without parameter
```

```
let
    Func = () => "sample function without parameter"
in
    Func()
```

Optional Parameters
parameters are defined as Required by default. that means you should specify the parameter at the time of invoking function. So if you want to define an optional parameter, use the Optional keyword as below:

let
Func = (optional x) => if x is null then "nothing" else x
in
Func(5)

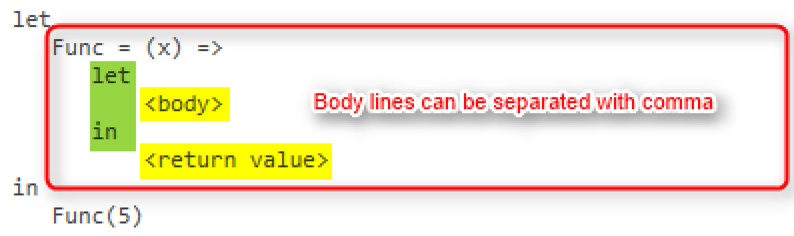


```
5
```

```
let
    Func = (optional x) => if x is null then "nothing" else x
in
    Func(5)
```

Variables in Function
samples above showed how to write single line expression function, but in most of the cases you would require to write multiple line function that contains variables inside the function. in that case you can define the function within the LET / IN structure as below:

```
let
  Func = (x) =>
    let
       <body>
    in
    <return value>
in
  Func(5)
```



```
let
   Func = (x) =>
      let
         <body>
      in
         <return value>
in
   Func(5)
```

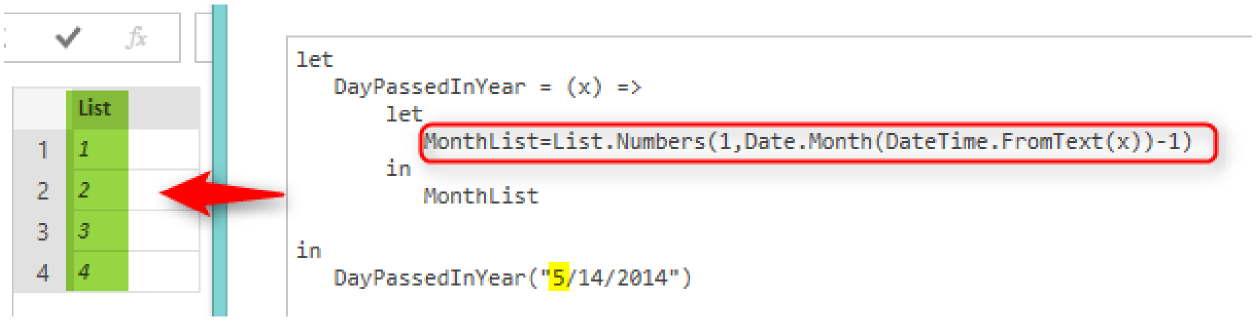
As you see in the above script another set of Let/In added inside the function. you can write body of function in the LET clause. and if you want to add multiple lines there, or if you want to define variables, do this as usual with a single comma at the end of each line. Finally, you can return value in the IN clause.

Example
following example show how we can use structure above to create a function that return number of days passed till a date from start of the year.

```
let
  DayPassedInYear = (x) =>
    let
      MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),
      Year=Date.Year(DateTime.FromText(x)),
      DaysInMonthList=List.Transform(MonthList,
         each Date.DaysInMonth(
            DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")
         )
       )
    in
      Date.Day(DateTime.FromText(x))+List.Sum(DaysInMonthList)
in
  DayPassedInYear("5/14/2014")
```

We used Let to define a multi-line body. the first line of body is defining a variable that creates list of Month numbers from the first month (1) to the previous month of the input date. for calculating the previous month of the input date we used this expression: *Date.Month(DateTime.FromText(x))-1* .
and this part generates a list based on numbers from a beginning number (1) to ending number (which would be result of previous month function): *List.Numbers(1,....)*

So as a result, the MonthList would be a variable that contains list of months from the first month up to prior month in the current year. Let's see how that single line would return result:



	List
1	1
2	2
3	3
4	4

```
let
   DayPassedInYear = (x) =>
        let
           MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1)
        in
           MonthList

in
   DayPassedInYear("5/14/2014")
```

Next line in the script calculates year of the input date and store that into a variable named "Year":

Year=Date.Year(DateTime.FromText(x)),
Consider that you should end each line with a single comma (if you don't want to logic of two line to be parsed together).

The final line of the body combined from multiple expressions, I'll describe the one by one;

DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")
Above expression will return first day of a month. Please note that there is a single underscore in the expression. that underscore would be used in EACH expression. EACH is a single parameter function. we used EACH in this example to apply a transformation to every item in the list. actually we want to replace each month number in the list, with the number of days in that month. So we use EACH single liner function to fetch number of days in that month. when you use EACH, you can use underscore as the parameter marker. In simpler words if you have a list as below:

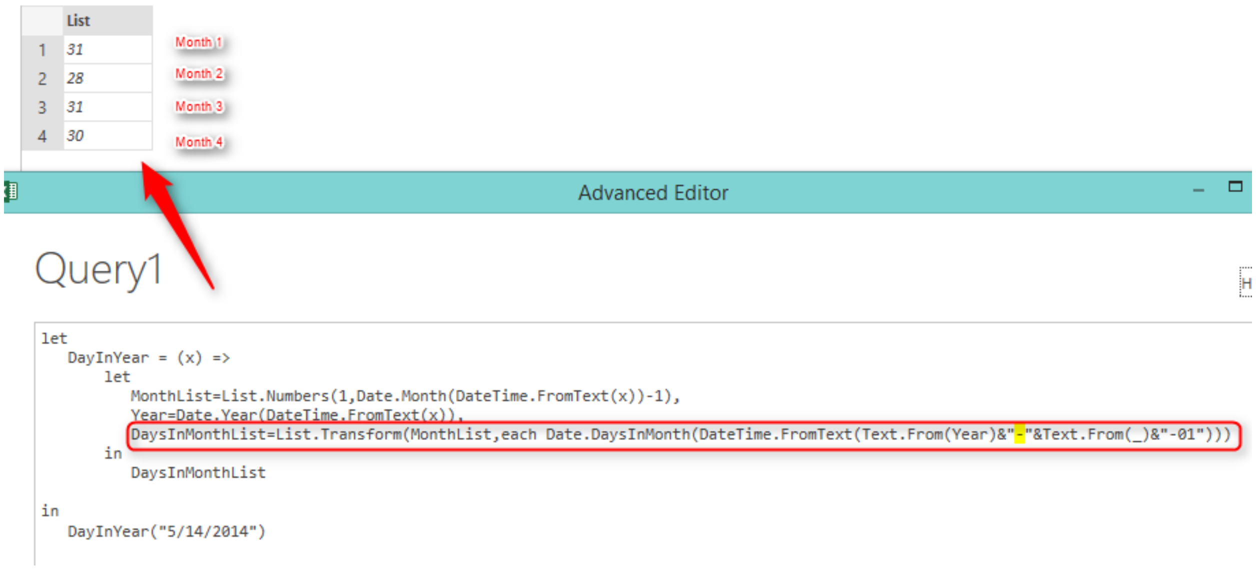
ListA
1
2
3
and if you transform that list with List.Transform function as below:

*List.Transform(ListA,each _*10)*
Result would be:

ListA
10
20
30

So, the result of This line below:
DaysInMonthList=List.Transform(MonthList,each Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01"))

Would be:



	List
1	31
2	28
3	31
4	30

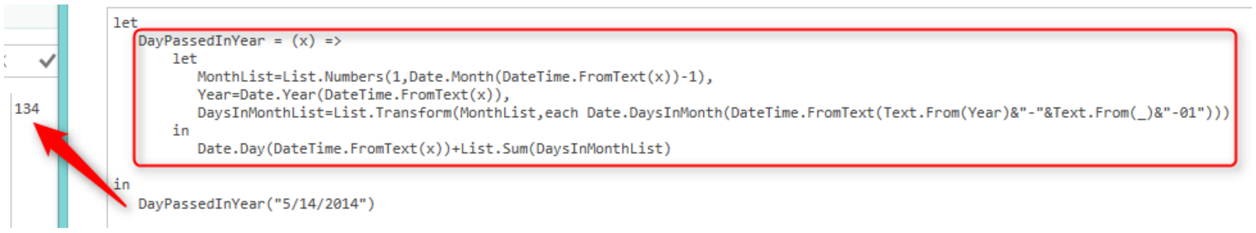
```
let
   DayInYear = (x) =>
       let
          MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),
          Year=Date.Year(DateTime.FromText(x)),
          DaysInMonthList=List.Transform(MonthList,each Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")))
       in
          DaysInMonthList

in
   DayInYear("5/14/2014")
```

As you see each list item which had the month number earlier, now replaced (transformed) with number of days in that month.

In the final step of the function, we return the result;
in
Date.Day(DateTime.FromText(x))+List.Sum(DaysInMonthList)

IN clause used to return the result, and we use Date.Day function to return the day part of input date. and we add that with sum of all values form the list. as a result, we would see number of days passed from the first of year to the input date.



```
let
   DayPassedInYear = (x) =>
       let
          MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),
          Year=Date.Year(DateTime.FromText(x)),
          DaysInMonthList=List.Transform(MonthList,each Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")))
       in
          Date.Day(DateTime.FromText(x))+List.Sum(DaysInMonthList)
in
   DayPassedInYear("5/14/2014")
```

Summary
In this lab, you've learned how to define functions and how to invoke them. you've learned that you can define optional or required parameters. you can also define multi-line functions that contains variables inside the body. you've seen a sample function that shows how useful are functions in real world. you've also learned about *each* keyword which can be used as a single parameter function (especially in lists and tables).