Power Query (M): Constants and curried custom IF functions

There are 6 documented constants in M

Name	Description
Number.E	Returns 2.7182818284590451, the value of e up to 16 decimal digits.
Number.Epsilon	Returns the smallest possible number.
Number.NaN	Represents 0/0.
Number.NegativeInfinity	Represents -1/0.
Number.PI	Returns 3.1415926535897931, the value for Pi up to 16 decimal digits.
Number.PositiveInfinity	Represents 1/0.

Let’s look at three of them

Name	Description
Number.E	Returns 2.7182818284590451, the value of e up to 16 decimal digits.
Number.Epsilon	Returns the smallest possible number.
Number.NaN	Represents 0/0.
Number.NegativeInfinity	Represents -1/0.
Number.PI	Returns 3.1415926535897931, the value for Pi up to 16 decimal digits.
Number.PositiveInfinity	Represents 1/0.

The result of division by zero depends on the numerator

```
= Table.AddColumn(AsTable,"Div",each [a]/[b])
```

	a	b	Div
1	-10	0	-Infinity
2	-9	0	-Infinity
3	-8	0	-Infinity
4	-7	0	-Infinity
5	-6	0	-Infinity
6	-5	0	-Infinity
7	-4	0	-Infinity
8	-3	0	-Infinity
9	-2	0	-Infinity
10	-1	0	-Infinity
11	0	0	NaN
12	1	0	Infinity
13	2	0	Infinity
14	3	0	Infinity
15	4	0	Infinity
16	5	0	Infinity
17	6	0	Infinity
18	7	0	Infinity
19	8	0	Infinity
20	9	0	Infinity
21	10	0	Infinity

Dividing a negative number by zero gives us *-Infinity*

Dividing zero by zero gives us *NaN*

Dividing a positive number by zero gives us *Infinity*

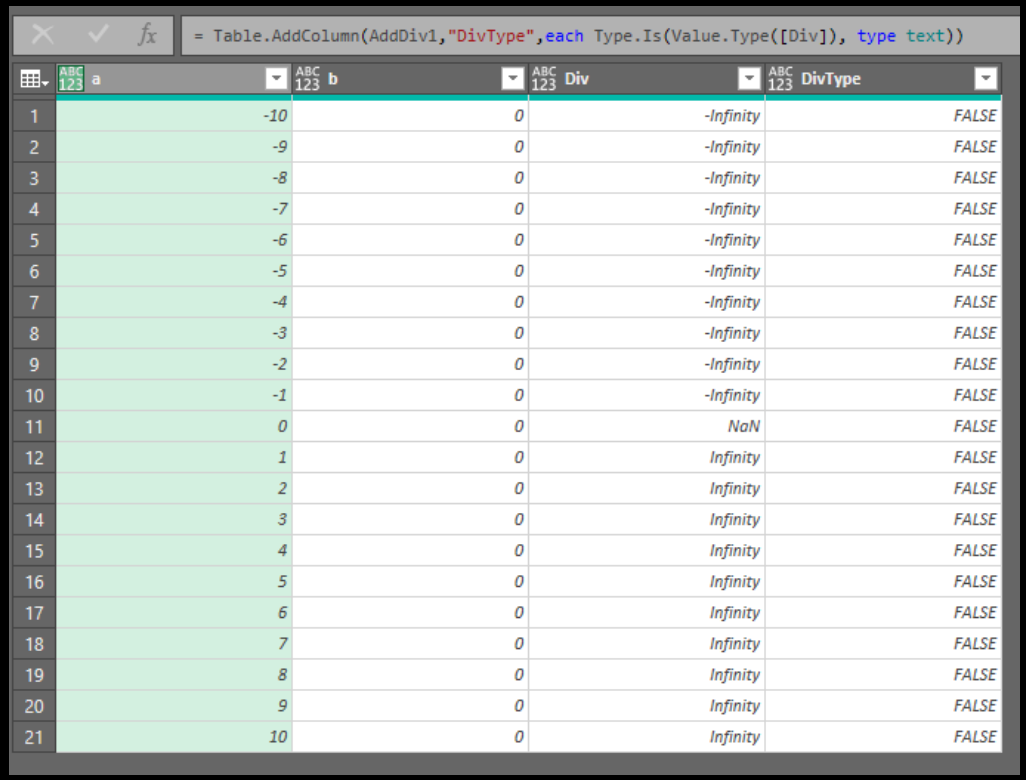
Though they look like text, they aren't

```
let
    a = {-10..10},
    b = List.Repeat({0},21),
    AsTable = Table.FromColumns({a,b},{"a","b"}),
    AddDiv = Table.AddColumn(AsTable,"Div",each [a]/[b]),
    AddDivType = Table.AddColumn(AddDiv,"DivType",each Type.Is(Value.Type([Div]), type text)),

```

Type.Is tests if the types of its two parameters are the same

Value.Type returns the type of the value in its parameter. Here, it gives us the type of the values in the [Div] column of the table



	a	b	Div	DivType
1	-10	0	-Infinity	FALSE
2	-9	0	-Infinity	FALSE
3	-8	0	-Infinity	FALSE
4	-7	0	-Infinity	FALSE
5	-6	0	-Infinity	FALSE
6	-5	0	-Infinity	FALSE
7	-4	0	-Infinity	FALSE
8	-3	0	-Infinity	FALSE
9	-2	0	-Infinity	FALSE
10	-1	0	-Infinity	FALSE
11	0	0	NaN	FALSE
12	1	0	Infinity	FALSE
13	2	0	Infinity	FALSE
14	3	0	Infinity	FALSE
15	4	0	Infinity	FALSE
16	5	0	Infinity	FALSE
17	6	0	Infinity	FALSE
18	7	0	Infinity	FALSE
19	8	0	Infinity	FALSE
20	9	0	Infinity	FALSE
21	10	0	Infinity	FALSE

We can use the constants to test the values

```
8     AddTestPositiveInfinity = Table.AddColumn(AddDiv,"TestPositiveInfinity",each [Div] = Number.PositiveInfinity),
9
10    AddTestNegativeInfinity = Table.AddColumn(AddTestPositiveInfinity,"TestNegativeInfinity",each [Div] = Number.NegativeInfinity),
```

	a	b	Div	TestPositiveInfinity	TestNegativeInfinity
1	-10	0	-Infinity	FALSE	TRUE
2	-9	0	-Infinity	FALSE	TRUE
3	-8	0	-Infinity	FALSE	TRUE
4	-7	0	-Infinity	FALSE	TRUE
5	-6	0	-Infinity	FALSE	TRUE
6	-5	0	-Infinity	FALSE	TRUE
7	-4	0	-Infinity	FALSE	TRUE
8	-3	0	-Infinity	FALSE	TRUE
9	-2	0	-Infinity	FALSE	TRUE
10	-1	0	-Infinity	FALSE	TRUE
11	0	0	NaN	FALSE	FALSE
12	1	0	Infinity	TRUE	FALSE
13	2	0	Infinity	TRUE	FALSE
14	3	0	Infinity	TRUE	FALSE
15	4	0	Infinity	TRUE	FALSE
16	5	0	Infinity	TRUE	FALSE
17	6	0	Infinity	TRUE	FALSE
18	7	0	Infinity	TRUE	FALSE
19	8	0	Infinity	TRUE	FALSE
20	9	0	Infinity	TRUE	FALSE
21	10	0	Infinity	TRUE	FALSE

In the case of *–Infinity* and *Infinity*, we can simply compare the value in the column with the constant using the equals operator

Testing for NaN is slightly different

```
11      AddTestNaNWithEquals = Table.AddColumn(AddDiv,"TestNaNWithEquals",each [Div] = Number.NaN),
12
13      AddTestNaNWithIsNaN = Table.AddColumn(AddTestNaNWithEquals,"TestNaNWithIsNaN",each Number.IsNaN([Div])),
```

	a	b	Div	TestNaNWithEquals	TestNaNWithIsNaN
1	-10	0	-Infinity	FALSE	FALSE
2	-9	0	-Infinity	FALSE	FALSE
3	-8	0	-Infinity	FALSE	FALSE
4	-7	0	-Infinity	FALSE	FALSE
5	-6	0	-Infinity	FALSE	FALSE
6	-5	0	-Infinity	FALSE	FALSE
7	-4	0	-Infinity	FALSE	FALSE
8	-3	0	-Infinity	FALSE	FALSE
9	-2	0	-Infinity	FALSE	FALSE
10	-1	0	-Infinity	FALSE	FALSE
11	0	0	NaN	FALSE	TRUE
12	1	0	Infinity	FALSE	FALSE
13	2	0	Infinity	FALSE	FALSE
14	3	0	Infinity	FALSE	FALSE
15	4	0	Infinity	FALSE	FALSE
16	5	0	Infinity	FALSE	FALSE
17	6	0	Infinity	FALSE	FALSE
18	7	0	Infinity	FALSE	FALSE
19	8	0	Infinity	FALSE	FALSE
20	9	0	Infinity	FALSE	FALSE
21	10	0	Infinity	FALSE	FALSE

To test for NaN, we must use the Number.IsNaN function

We can test for invalid numbers with a custom function

```
12      fnInvalidNumber = (n as number) as logical
13          => n = Number.PositiveInfinity or n = Number.NegativeInfinity or Number.IsNaN(n),
14
15      AddTestWithfnInvalidNumber = Table.AddColumn(AddDiv,"TestWithfnInvalidNumber",each fnInvalidNumber([Div])),
```

	a	b	Div	TestWithfnInvalidNumber
1	-10	0	-Infinity	TRUE
2	-9	0	-Infinity	TRUE
3	-8	0	-Infinity	TRUE
4	-7	0	-Infinity	TRUE
5	-6	0	-Infinity	TRUE
6	-5	0	-Infinity	TRUE
7	-4	0	-Infinity	TRUE
8	-3	0	-Infinity	TRUE
9	-2	0	-Infinity	TRUE
10	-1	0	-Infinity	TRUE
11	0	0	NaN	TRUE
12	1	0	Infinity	TRUE
13	2	0	Infinity	TRUE
14	3	0	Infinity	TRUE
15	4	0	Infinity	TRUE
16	5	0	Infinity	TRUE
17	6	0	Infinity	TRUE
18	7	0	Infinity	TRUE
19	8	0	Infinity	TRUE
20	9	0	Infinity	TRUE
21	10	0	Infinity	TRUE

And use a custom IF function to abstract the replacement with a default value

```
17    fnIF = (testValue as any, withFunction as function, whenTrue as any, optional whenFalse as any) as any
18        => let
19                _whenFalse = if whenFalse is null then testValue else whenFalse
20            in
21                if withFunction(testValue) then whenTrue else _whenFalse,
22
```

"Test the value testValue with the function withFunction.

If withFunction returns true, return whenTrue.

If withFunction returns false, return whenFalse if provided, testValue otherwise"

We can make this more robust by adding logic to make sure withFunction returns a logical value

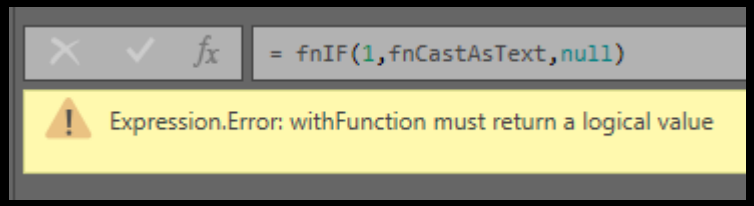
Type.FunctionReturn retrieves the return type of the function type in its parameter

```
16  fnIF = (testValue as any, withFunction as function, whenTrue as any, optional whenFalse as any) as any
17      => let
18              withFunctionReturnType = Type.FunctionReturn(Value.Type(withFunction)),
19
20              withFunctionIsLogical = Type.Is(withFunctionReturnType,type logical),
21
22              _whenFalse = if whenFalse is null then testValue else whenFalse
23          in
24              if withFunctionIsLogical
25              then
26                  if withFunction(testValue) then whenTrue else _whenFalse
27              else error "withFunction must return a logical value",
```

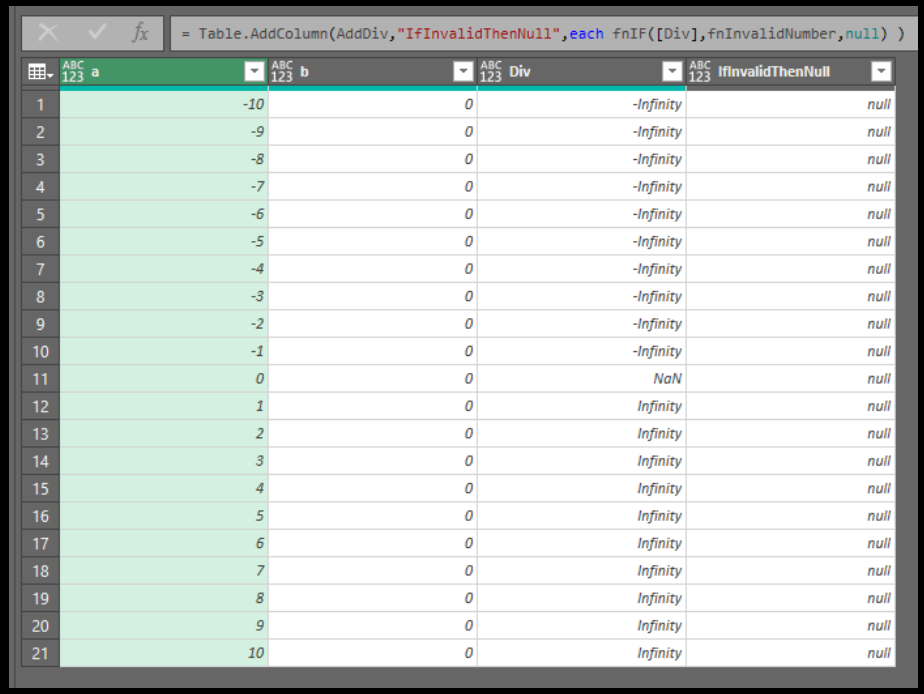
We can then test if that return type is of *type logical* using Type.Is

And add logic to return the result of the original expression if withFunction is a logical function or a custom error if it isn't

fnIF can then be used with any logical function



fnIF returns an error if the function passed to the second parameter returns anything other than a logical value



	a	b	Div	IfInvalidThenNull
1	-10	0	-Infinity	null
2	-9	0	-Infinity	null
3	-8	0	-Infinity	null
4	-7	0	-Infinity	null
5	-6	0	-Infinity	null
6	-5	0	-Infinity	null
7	-4	0	-Infinity	null
8	-3	0	-Infinity	null
9	-2	0	-Infinity	null
10	-1	0	-Infinity	null
11	0	0	NaN	null
12	1	0	Infinity	null
13	2	0	Infinity	null
14	3	0	Infinity	null
15	4	0	Infinity	null
16	5	0	Infinity	null
17	6	0	Infinity	null
18	7	0	Infinity	null
19	8	0	Infinity	null
20	9	0	Infinity	null
21	10	0	Infinity	null

And concisely returns null if the result of the test of [Div] using fnInvalidNumber is *true*

Further to this, we can curry fnIF if we expect certain functions to be used repeatedly

fnIF takes a single parameter of type function

```
23  fnIF = (withFunction as function) =>
24      let
25          withFunctionReturnType = Type.FunctionReturn(Value.Type(withFunction)),
26          withFunctionIsLogical = Type.Is(withFunctionReturnType,type logical)
27      in
28          if withFunctionIsLogical
29          then
30              (testValue as any, whenTrue as any, optional whenFalse as any) as any
31                  => let
32                          _whenFalse = if whenFalse is null then testValue else whenFalse
33                      in
34                          if withFunction(testValue) then whenTrue else _whenFalse
35          else error "withFunction must return a logical value",
36
```

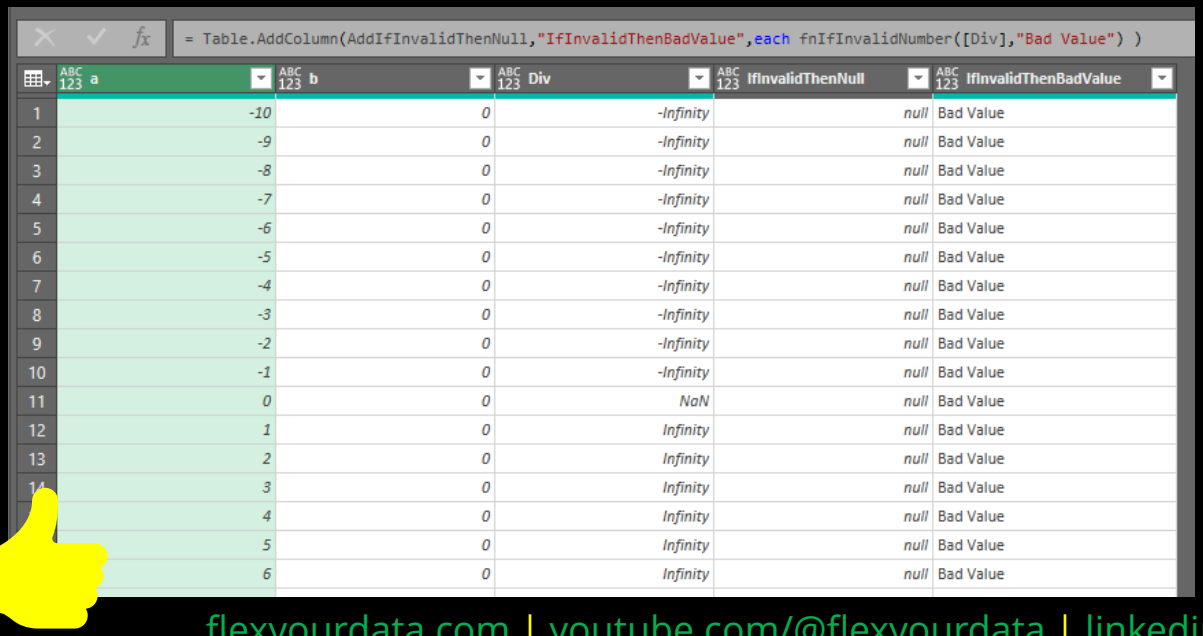
The type checking logic happens up-front

The return value is a simpler function for:

if-then-else-with-default

And we can create separated fnIF versions using different withFunctions

```
37
38  //fnIfInvalidNumber = (testValue as any, whenTrue as any, optinal whenFalse as any) as any
39  fnIfInvalidNumber = fnIF(fnInvalidNumber),
40
41  //fnIfSomeOtherTest = (testValue as any, whenTrue as any, optinal whenFalse as any) as any
42  fnIfSomeOtherTest = fnIF(fnSomeOtherTest),
43
44  AddIfInvalidThenNull
45      = Table.AddColumn(AddDiv,"IfInvalidThenNull",each fnIfInvalidNumber([Div],null) ),
46
47  AddIfInvalidThenBadValue
48      = Table.AddColumn(AddIfInvalidThenNull,"IfInvalidThenBadValue",each fnIfInvalidNumber([Div],"Bad Value") ),
```



= Table.AddColumn(AddIfInvalidThenNull,"IfInvalidThenBadValue",each fnIfInvalidNumber([Div],"Bad Value"))

	a	b	Div	IfInvalidThenNull	IfInvalidThenBadValue
1	-10	0	-Infinity	null	Bad Value
2	-9	0	-Infinity	null	Bad Value
3	-8	0	-Infinity	null	Bad Value
4	-7	0	-Infinity	null	Bad Value
5	-6	0	-Infinity	null	Bad Value
6	-5	0	-Infinity	null	Bad Value
7	-4	0	-Infinity	null	Bad Value
8	-3	0	-Infinity	null	Bad Value
9	-2	0	-Infinity	null	Bad Value
10	-1	0	-Infinity	null	Bad Value
11	0	0	NaN	null	Bad Value
12	1	0	Infinity	null	Bad Value
13	2	0	Infinity	null	Bad Value
14	3	0	Infinity	null	Bad Value
	4	0	Infinity	null	Bad Value
	5	0	Infinity	null	Bad Value
	6	0	Infinity	null	Bad Value

Takeaways

1. Some division operations can result in special values being returned

2. These can be tested with the constants Number.PositiveInfinity, Number.NegativeInfinity and the function Number.IsNaN

3. The common pattern of *If (result of test on value) then default else value* can be abstracted and applied consistently using custom functions