ChatGPT

Executive Summary

Power Query M bietet die Möglichkeit, eigene *benutzerdefinierte Funktionen* zu erstellen, um wiederkehrende Transformationen und Berechnungen zu kapseln und in verschiedenen Abfragen wiederzuverwenden. Eine Funktion in M ist dabei **eine Abbildung von Eingabewerten auf einen einzelnen Ausgabewert** [1]. Die M-Sprache definiert Funktionen mit der Syntax:

```
(<Parameter1> as <Typ1>, <Parameter2> as <Typ2>, …) as <Rückgabetyp> =>
<Funktionskörper>
```

Zum Beispiel:

```
let
  AddOne = (x as number) as number => x + 1,
  Result = AddOne(5)
in
  Result
```

gibt `6` zurück [2]. Ist der Rückgabetyp angegeben, erzwingt M die Übergabe dieses Datentyps. Optional können Parameter als `optional` markiert werden, was fehlende Argumente in `null` umwandelt [3].

In dieser Anleitung werden die Grundlagen erläutert, und sukzessive fortgeschrittene Konzepte eingeführt. Es werden praxisnahe Beispiele zu Text- und Zahlenmanipulation, Listengeneratoren, rekursiven Funktionen und mehr gezeigt. Wir diskutieren auch, wie mehrere Funktionen in Bibliotheken organisiert, dokumentiert und versionskontrolliert werden können. Zudem beleuchten wir das Teilen von Funktionen zwischen Arbeitsmappen (z.B. über Dataflows) sowie praktische Anwendungsszenarien (z.B. Datenbereinigung). Abschließend finden Sie Übungen für verschiedene Level, sowie einen Troubleshooting-Guide und weiterführende Ressourcen.

1. Grundlagen

1.1 Eine leere Abfrage zur Funktion machen

In Power Query (Excel oder Power BI) beginnen Sie mit **einer leeren Abfrage**. Dazu im Abfrage-Editor mit Rechtsklick auf „Abfragen“ eine *Leere Abfrage* einfügen. Im **Erweiterten Editor** sehen Sie standardmäßig ein `let … in …`-Gerüst. Dieses überschreiben Sie durch Ihren Funktionscode. Beispiel:

```
let
    MeineFunktion = () => "Hallo Welt"
```

```
in
    MeineFunktion
```

Hier definiert `MeineFunktion` eine Funktion ohne Parameter, die stets den Text *"Hallo Welt"* zurückgibt. Nach **Fertigstellen** erscheint in der Abfrageliste ein Fx-Symbol – Ihre Query wurde zur Funktion. Sie können sie nun über „Aufrufen“ oder in Code mit `MeineFunktion()` ausführen [4] [5].

Beispiel: Funktion mit Parametern

Fügen Sie in die Klammern Parameter und Typen ein. Beispiel: Funktion zum Addieren zweier Zahlen:

```
let
  AddiereZwei = (a as number, b as number) as number => a + b,
  Test = AddiereZwei(3, 7)
in
  Test
```

Hier gibt **AddiereZwei** den Summenwert zurück. `Test` wird zu `10`. Der linke Teil vor `=>` listet die Parameter, der rechte Teil ist der Ausdruck. Dieses Muster ist typisch für M-Funktionen [2]. Ohne Angabe des Rückgabetyps (z.B. bei `(a,b) => a+b`) behandelt M die Parameter und Rückgabe als „any“-Typ [6].

1.2 Syntax und Konventionen

- **Function Arrow (=>)**: In Power Query leitet man Funktionen mit dem Pfeilsymbol `=>` ein [5]. Alles vor dem `=>` sind Parameter (mit Typen), alles danach der Funktionskörper. Beispiel mit Parameter und Datentyp:

```
let
  fnBegruessung = (name as text) as text => "Hallo, " & name
in
  fnBegruessung
```

- **Namenskonventionen**: Funktionennamen und Parameter sollten beschreibend sein. In vielen Projekten wird CamelCase oder Unterstrich-Schreibweise verwendet (z.B. `fnAddiereZahl` oder `addiereZahl`). Parameter beginnen oft kleingeschrieben (z.B. `text`, `zahl`). Wichtig ist Konsistenz und Klarheit. Achten Sie darauf, Funktionennamen am Ende der Abfrage (nach dem `in`) zurückzugeben.

- **Parameterdefinition**: Innerhalb der Klammern definieren Sie *Pflicht-Parameter* mit `as Typ`. Z.B. `x as number`. Optional-Parameter markieren Sie mit `optional` [3]. Beispiel für einen optionalen Parameter `y`:

```
(x as number, optional y as number) as number =>
  if y = null then x else x + y
```

Wird `y` nicht übergeben, erhält `y` den Wert `null` [3].

- **Rückgabetyp**: Schreiben Sie `as Typ` direkt vor `=>` (außerhalb der Klammer) zur Spezifikation des Rückgabetyps [6]. Dadurch wird bei falschem Ausgabewert ein Fehler generiert. Beispiel `...) as number => ...`.

1.3 Einfache Beispiele

Textmanipulation

M bietet eingebaute Textfunktionen. Beispiel: Funktion zum Aufteilen eines Strings in eine Liste von Worten:

```
let
  fnSplitWords = (text as text) as list => Text.Split(text, " "),
  Result = fnSplitWords("Power Query M")
in
  Result
```

Ergebnis: `{ "Power", "Query", "M" }`. (Hinweis: die Spracheinstellung kann `;` statt `,` erfordern.) Solche Funktionen können Sie über `Table.TransformColumns` auf ganze Spalten anwenden.

Mathematische Operationen

Zum Beispiel eine einfache Addition oder eine Funktion für den Zinseszins:

```
let
  fnZinseszins = (kapital as number, faktor as number, jahre as number) as
number =>
    kapital * Number.Power(faktor, jahre),
  Result = fnZinseszins(1000, 1.05, 5)
in
  Result
```

Hier liefert die Funktion den Endwert nach 5 Jahren zu 5 % Zinsen.

Fehlerbehandlung

Sie können `try ... otherwise` oder `try ... catch` verwenden, um Fehler innerhalb einer Funktion abzufangen [7]. Beispiel: Sichere Division, die bei Division durch Null eine Meldung ausgibt:

```
let
    SafeDivide = (x as number, y as number) as text =>
        try Text.From(x / y) otherwise "Fehler: Division durch 0",
    Positiv = SafeDivide(10, 2),
    Negativ = SafeDivide(5, 0)
```

```
in
    [Ergebnis1 = Positiv, Ergebnis2 = Negativ]
```

Hier gibt `Positiv` `"5"` und `Negativ` `"Fehler: Division durch 0"` zurück. Diese `try`-Konstruktion verhindert, dass ein Fehler Ihre Abfrage bricht, und erlaubt stattdessen eine definierte Behandlung [7].

1.4 Zusammenfassung Grundlagen

Power Query-Funktionen beginnen also meist mit `let <Funktion> = (...) => ... in <Funktion>`. Die Parameter und ihr Typ stehen links vom `=>`, der Ausdruck rechts. Einfache Beispiele haben wir oben gezeigt. Durch Angabe von `as <Typ>` und `optional` steuern Sie Anforderungen an Eingaben. Mit `try ... otherwise` fangen Sie Fehler ab [7]. Diese Basics sind die Grundlage für weitergehende Techniken.

2. Fortgeschrittene Funktionen

2.1 Bedingte Logik und Schleifen

If-Then-Else

Für komplexe Logik verwenden Sie verschachtelte `if ... then ... else`. Zum Beispiel eine Funktion, die je nach Parameter positiv oder negativ multipliziert:

```
let
  fnSign = (wert as number, positiv as logical) as number =>
    if positiv then wert else -wert,
  Positiv = fnSign(10, true),
  Negativ = fnSign(10, false)
in
  [Positiv = Positiv, Negativ = Negativ]
```

Schleifen-Ähnliches mit List.Generate

Power Query hat keine klassische Schleife wie `for`, aber **List.Generate** ermöglicht iterative Erzeugung von Listen [8]. Beispiel: Erstelle eine absteigende Liste von 5:

```
let
  Countdown = List.Generate(
      () => 5,              // Startwert
      each _ > 0,           // Bedingung (solange >0)
      each _ - 1            // Nächster Wert
  )
in
  Countdown
```

Ergebnis: `{5, 4, 3, 2, 1}` [9]. `List.Generate` benötigt eine Funktion für Startwert, eine Prüfbedingung und eine Transformationsfunktion.

List.Accumulate (Akkumulieren)

Mit **List.Accumulate** können Sie über eine Liste aggregieren [10]. Beispiel: Summe einer Liste:

```
let
  Summe = List.Accumulate({1,2,3,4,5}, 0, (state, aktueller) => state +
aktueller)
in
  Summe
```

Ergebnis: `15` [11]. Der zweite Parameter `0` ist der Startwert (`state`), die Lambda-Funktion addiert nacheinander alle Listenelemente.

2.2 Listen, Records und Tabellen

Power Query-Funktionen können **Nicht-Skalare** Werte zurückgeben. Neben einzelnen Werten lassen sich **Listen**, **Datensätze (Records)** oder **Tabellen** als Ergebnis definieren [12].

- **Listen:** Verwenden Sie z.B. `List.Transform` oder `List.Generate`. Eine Funktion könnte etwa alle Quadratzahlen in einer Liste liefern. Beispiel:

```
let
  fnQuadrate = (n as number) as list => List.Transform({1..n}, each _ * _),
  QuadratBis5 = fnQuadrate(5)
in
  QuadratBis5  // {1,4,9,16,25}
```

- **Records:** Nützlich, um mehrere benannte Werte zurückzugeben. Beispiel:

```
let
  fnPerson = (name as text, alter as number) as record => [Name = name, Alter
= alter],
  Person = fnPerson("Anna", 30)
in
  Person  // [Name="Anna", Alter=30]
```

- **Tabellen:** Sehr mächtig! Sie können z.B. eine Tabelle aus einer Liste erstellen oder übergeben und mit `Table.FromList`, `Table.AddColumn` etc. bearbeiten. Beispiel:

```
let
  fnListeZuTabelle = (lst as list) as table =>
    Table.FromList(lst, Splitter.SplitByNothing(), {"Wert"}),
  Tabelle = fnListeZuTabelle({10,20,30})
```

```
in
  Tabelle
```

Erzeugt eine Tabelle mit einer Spalte „Wert“ und den Zahlen 10,20,30.

Diese Vielseitigkeit erlaubt komplexe Berechnungen. In der Praxis können Sie z.B. eine Funktion entwickeln, die Daten aus verschiedenen Quellen liest und als kombinierte Tabelle zurückgibt.

2.3 Rekursive Funktionen

M erlaubt Selbstaufruf von Funktionen mit dem **@-Operator** [13] . Damit lassen sich klassische rekursive Algorithmen implementieren. Beispiel: Fakultät:

```
let
  fact = (num as number) as number =>
      if num = 0 then 1 else num * @fact(num - 1),
  Result = fact(5)
in
  Result
```

Ergebnis: `120` (Fakultät von 5) [13] . Der `@` vor `fact` stellt sicher, dass innerhalb des Funktionskörpers auf die Funktion selbst verwiesen wird. **Achtung:** Rekursion kann sehr rechenintensiv sein. Verwenden Sie sie nur, wenn nötig. Oft lässt sich dasselbe Ergebnis effizienter mit Listen- oder Akkumulationsfunktionen erzielen [14] .

2.4 Performance-Optimierung

- **Table.Buffer nutzen:** Wenn Sie eine Funktion haben, die auf großen Tabellen mehrfach zugreift, kann es sinnvoll sein, die Tabelle einmal zu puffern. Beispiel:

  „Wenn Sie keine andere Möglichkeit haben, die Vorlage jedes Mal zu laden, könnte `Table.Buffer` entscheidend sein, um die Ladezeit zu reduzieren (statt sie 4 × 263 Mal zu wiederholen) [15] .“

Mit `Table.Buffer` wird eine Tabelle im Speicher zwischengespeichert, so dass wiederholte Zugriffe schneller erfolgen.

- **Query Folding beachten:** Generell gilt, teure Operationen (Sortieren, Gruppieren) möglichst spät in den Abfragen zu machen und so viel wie möglich auf Quellenebene zu filtern (Query Folding) [16] .

- **Vereinfachung:** Vermeiden Sie aufwändige Funktionen innerhalb von Zeilenschleifen. Beispiel: Eine benutzerdefinierte Funktion sollte nicht pro Zeile auf das ganze Dataset zugreifen. Versuchen Sie stattdessen, Tabellentransformationen (z.B. `Table.AddColumn` mit eingebauter Logik) oder Listenoperationen (wie oben) zu verwenden, um Performance-Einbußen zu vermeiden.

3. Funktions-Bibliotheken

3.1 Organisation und Kategorisierung

In größeren Projekten sammelt man oft viele Funktionen. Organisieren Sie diese in **Gruppen** oder **sekundären Dateien**. In Power BI/Excel können Sie alle Funktionen in eine separate Abfrage (z.B. „Funktionen“) packen und dort untereinander als einzelne Queries anlegen [17]. Jede Query ist dabei eine Funktion, deren Name als Abfrage-Name gewählt wird.

Alternativ können Sie in einem dedizierten „Funktions-PBIX“ oder einer Excel-Datei sämtliche M-Funktionen ablegen und bei Bedarf per `Power Query -> Abfragen abrufen` importieren. Auch der Einsatz von **Power Query Dataflows** (in Power BI Service) bietet sich an: Ein Dataflow kann zahlreiche Funktionen enthalten, die dann von verschiedenen Berichten durch Abfrage der Dataflow-Entitäten wiederverwendet werden [18].

3.2 Dokumentation und Teststrategie

Dokumentieren Sie Ihre Funktionen, um sie für andere Team-Mitglieder verständlich zu machen:

- **Kommentarfelder und Metadaten:** Es gibt fortgeschrittene Möglichkeiten, Metadaten in M zu verwenden. Beispielsweise lassen sich über die `Documentation.*`-Felder in `Value.ReplaceType` Beschreibungen, Namen und Beispiel-Codes zu Funktionen hinzufügen [19] [20]. Dies wird vor allem bei komplexen *Custom Connectors* genutzt, wirkt sich aber auch in Power BI aus, wenn Funktionen über das UI aufgerufen werden (anzeigen von Beschriftungen und Hilfetexten). Beispiel (nur Ausschnitt):

```
shared HelloWorld =
  Value.ReplaceType(
    (msg as (type text meta [
        Documentation.FieldCaption = "Nachricht",
        Documentation.FieldDescription = "Anzuzeigender Text"
    ]))
    as table meta [
      Documentation.LongDescription = "Gibt die Nachricht als Tabelle zurück"
    ],
    HelloWorldType
  )
```

- **Kurze Handbücher:** Listen Sie Funktion, Parameter und Beispiele in einem separaten Dokument auf. Alternativ können Sie in einem Readme oder Wiki für Ihre Team-Bibliothek kurze Code-Beispiele geben.

- **Testen:** Testen Sie Funktionen mit verschiedenen Eingaben, bevor Sie sie produktiv verwenden. Erstellen Sie dazu kleine Abfragen, die Ihre Funktion aufrufen (Invoke) und vergleichen Sie die Rückgaben mit erwarteten Ergebnissen. So finden Sie Fehler frühzeitig. Für besonders kritische Module kann man sogar *Unit Tests* verwenden: Im Power Query SDK (Visual Studio) lassen sich automatisierte Tests definieren [21], die den erwarteten Output prüfen.

3.3 Versionskontrolle und Team-Sharing

Power Query M Code kann nicht direkt in Git verwaltet werden, da er in PBIX/XLSX eingebettet ist. Lösungen dafür sind:

- **.pq-Dateien in Git:** Sie können Ihre M-Funktionen in reinen Textdateien (*.pq) speichern. Zum Beispiel kopieren Sie den Inhalt einer Funktion aus dem Advanced Editor und legen eine `.pq`-Datei an. Diese Versionieren Sie in Git wie üblichen Code [22].

- **Expression.Evaluate Trick:** Eine fortgeschrittene Methode ist es, Funktionen in einem Git-Repo (oder einem Dateipfad) zu speichern und sie zur Laufzeit zu laden [23] [24]. Beispiel:

```
let
  Source = Expression.Evaluate(
    Text.FromBinary(Web.Contents("https://raw.githubusercontent.com/Name/
Repo/master/meineFunktionen.pq")),
    #shared
  )
in
  Source
```

Dieser Code lädt den Inhalt der Datei `meineFunktionen.pq` von GitHub, wertet ihn als M-Code aus und gibt so die dort definierten Funktionen zurück [23]. Achtung: Solche dynamischen Ladevorgänge können Sicherheitsrisiken bergen (s.u.).

- **Team Sharing:** Im Team bieten sich zentrale Repositories oder Dataflows an. Wie oben erwähnt, lassen sich Funktionen in einem Dataflow veröffentlichen und von allen Kollegen „abonnieren“ [18]. Andernfalls stellt man regelmäßig Updates der Funktionsbibliothek bereit (z.B. per OneDrive/SharePoint-Datei oder in einem Version-Control-System) und bindet sie ein.

4. Cross-Workbook Modules

4.1 Wiederverwendbare Funktionbibliotheken über Arbeitsmappen

Power Query selbst kennt keine direkte *Importfunktion* von Funktionen zwischen Dateien. Strategien:

- **Template-Arbeitsmappe:** Legen Sie eine Excel-Datei (oder PBIX) nur mit Funktionen an. Weitere Reports oder Dateien können beim Einrichten diese Datei öffnen und die relevanten Funktionen kopieren. Power Query kann per `Datei > Daten abrufen > Aus Arbeitsmappe` eine Abfrage inkl. Funktion importieren.

- **Dataflows (Power BI Service):** Wie bereits angedeutet: Speichern Sie Funktionen in einem Dataflow, dann verwenden andere Berichte im Power BI Desktop „Get Data → Dataflows“, um diese Funktionen zu laden [25]. Dataflows bieten so eine *Server-seitige Bibliothek*.

- **Power BI Dataverse:** Unternehmen mit Microsoft Fabric/Power BI Premium können Funktionen auch in Dataverse oder durch Dataflows bereitstellen (z.B. ETL-Chaining, Computed Entities), womit sie zentral genutzt werden können [26].

- **Dependency Management:** Wenn Funktionen sich gegenseitig aufrufen oder von externen Daten abhängen (z.B. Webservices), dokumentieren Sie diese Abhängigkeiten. Verwenden Sie z.B. Namenskonventionen oder Metadaten, um Versionen anzugeben. Bei Dataflows kann das Deployment über Umgebungen (Dev/Test/Prod) gesteuert werden.

4.2 Import/Export von Funktionen

- **M Code Export:** Im Power Query Editor können Sie den reinen M-Code via „Erweiterten Editor" kopieren. Diesen z.B. in eine `.pq`-Datei einfügen oder in ein Ticketsystem/Git. Importieren geschieht analog – den Code in einen neuen Abfrage-Editor einkopieren.

- **Shared Keyword:** Innerhalb eines Dokuments liefert die spezielle Query `#shared` alle verfügbaren Funktionen (auch eigene) in einer Tabelle [27]. So können Kollegen mit dem UI nach Funktionen suchen. In externen Dateien hilft das allerdings nicht direkt, aber es ist ein Tipp, um im eigenen Bericht Funktionen schnell zu finden.

- **Enterprise-Deployment:** In großen Unternehmen kann man Power Query Funktionen als Teil eines Custom Data Connector (M-Extension) veröffentlichen und zentral warten. Dies setzt aber umfangreichere Tools (Power Query SDK) und administrative Freigaben voraus.

5. Praxisbeispiele

5.1 Datenbereinigung

- **Trimmen und Bereinigen:** Eine typische Funktion entfernt führende/folgende Leerzeichen, Zeilenumbrüche etc. Beispiel:

```
let
  fnCleanText = (t as text) as text => Text.Clean(Text.Trim(t)),
  Beispiel = fnCleanText("  Hallo \t\n")
in
  Beispiel
```

Dies ergibt `"Hallo"`. Eine solche Funktion kann auf Textspalten angewendet werden, um Tippfehler oder unsichtbare Zeichen zu eliminieren.

- **Fehlende Werte behandeln:** Funktion zum Ersetzen von `null` durch Standardwert:

```
let
  fnDefault = (wert as any, default as any) =>
     if wert = null then default else wert,
  Ausgef = fnDefault(null, "N/A")
```

```
in
  Ausgef
```

Gibt `"N/A"` zurück. Praktisch z.B. um Tabellen zu konsolidieren, wenn Spalten Lücken haben.

5.2 Komplexe Berechnungen

- **Index-/Kennzahlen:** Beispiel eine Funktion, die eine ID basierend auf Datum anfügt (vereinfachtes Beispiel ohne Sortierung):

```
let
  fnAddIndex = (tbl as table, indexName as text) as table =>
    Table.AddIndexColumn(tbl, indexName, 1, 1),
  DatumTabelle = #table({"Datum", "Wert"}, {{#date(2023,1,1), 100},
{#date(2023,1,2), 150}}),
  Ergebnis = fnAddIndex(DatumTabelle, "ID")
in
  Ergebnis
```

Fügten wir eine Spalte „ID“ hinzu: die Tabelle hat nun Datum, Wert, ID.

- **Validierung:** Prüfen, ob Daten das erwartete Format haben. Z.B. E-Mail-Validierung (einfach):

```
let
  fnValidateEmail = (email as text) =>
    if Text.Contains(email, "@") then email else error "Ungültige E-Mail",
  Test1 = try fnValidateEmail("max@muster.de") otherwise "fehlerhaft",
  Test2 = try fnValidateEmail("keine_at") otherwise "fehlerhaft"
in
  [Valid = Test1, Invalid = Test2]
```

`Test1` liefert `"max@muster.de"`, `Test2` `"fehlerhaft"`.

- **Berechnung komplexer Kennzahlen:** Stellen Sie sich vor, Sie müssen währungsbezogene Berechnungen oder Rangplätze ermitteln. Sie können dafür eigene Funktionen bauen, die Parameter entgegennehmen (z.B. Kurs, Skalierungsfaktor) und dann in Spaltenformel oder benutzerdefinierter Spalte aufgerufen werden.

5.3 Performance-Fallen und deren Vermeidung

- **Teure Funktionen pro Zeile:** Falls Sie in einer großen Tabelle für jede Zeile eine Funktion aufrufen (z.B. zum Berechnen einer Statistik), kann das sehr langsam sein. Besser ist es oft, Listen- oder Tabellenoperationen gebündelt anzuwenden, oder Zwischenergebnisse (mit Table.Buffer) zu speichern [15] .

- **Nichtgebufferte Vorlagen:** Wenn Ihre Funktion etwa eine Vorlage-Tabelle aus einer Datei mehrfach nutzt, puffern Sie diese (siehe oben). Ansonsten lädt Power Query die Datei bei jedem Funktionsaufruf erneut.

- **Vermeiden unnötiger Rekursion:** Tief rekursive Funktionen (z.B. auf großen Listen) können zu Stack-Overflow oder langen Ladezeiten führen. Prüfen Sie, ob eine iterative Listenerzeugung (List.Generate) schneller wäre.

6. Praktische Übungen

1. **Einsteiger:**
2. Schreiben Sie eine Funktion `fnAddiere`, die zwei Zahlen entgegennimmt und ihre Summe zurückgibt. Testen Sie sie mit verschiedenen Werten.
3. Erstellen Sie eine Funktion `fnGibDatum`, die das aktuelle Datum zurückliefert (ohne Parameter). Invoke im Editor und prüfen Sie das Ergebnis.
4. **Fortgeschritten:**
5. Bauen Sie eine Funktion `fnDatenReinigen`, die einem Textfeld Leerzeichen am Anfang/Ende und Zeilenumbrüche entfernt (verwenden Sie `Text.Trim` und `Text.Clean`). Wenden Sie die Funktion auf eine Tabellenspalte an.
6. Implementieren Sie mit `List.Generate` eine Funktion `fnFib`, die die ersten *n* Fibonacci-Zahlen erzeugt und als Liste zurückgibt.
7. **Experte:**
8. Erstellen Sie ein kleines Repository (Ordner oder Git) und speichern Sie dort einen Satz `.pq`-Dateien mit Funktionen (z.B. `fnSumme.pq`, `fnMax.pq`). Schreiben Sie eine Abfrage, die diese per `Expression.Evaluate` in Power Query lädt [23]. Achten Sie darauf, wie Sie mit dem `#shared` Kontext arbeiten.
9. Verfassen Sie für eine Ihrer Funktionen eine Dokumentation, indem Sie die `Documentation.FieldDescription`-Felder setzen, und prüfen Sie, wie sie im Power Query-Dialog angezeigt wird [19]. (Tipp: Dieselbe Funktion als *benannte Funktion* über die Benutzeroberfläche erstellen und dann in den Erweiterten Editor wechseln, um die Metadaten hinzuzufügen.)

7. Troubleshooting Guide

- **Syntaxfehler („Expression.Error"):** Prüfen Sie auf fehlende Kommas, Klammern oder Anführungszeichen. Achten Sie darauf, dass die `in <Name>`-Zeile den Bezeichner Ihrer Funktion oder eines definierten Schritts enthält. Beispiel: Nach `let ... in` muss der Name der Variable stehen, die zurückgegeben werden soll.
- **Typinkompatibilität:** Wenn Sie z.B. `as number` verwenden, muss der Ausdruck wirklich numerisch sein. Bei Konflikten werfen viele Funktionen Fehler mit Hinweisen wie *"expected number, found text"*. Nutzen Sie ggf. `try ... otherwise` oder Typkonvertierungen (`Number.FromText`, `Text.From`, etc.).
- **Funktionsaufruf funktioniert nicht:** Ist Ihre Funktion in einer anderen Abfrage definiert (als gemeinsame Abfrage) und heißt z.B. `fnFoo`, können Sie sie in einer Zeilentransformation mit der syntax `fnFoo([Feld1], [Feld2])` aufrufen. Wichtig: Die Funktionen müssen vorher als Queries existieren (mit fx-Symbol) [28]. In einem anderen PBIX geht das nur über Kopieren oder Dataflow.
- **Leere Ergebnisse / Null:** Prüfen Sie Ihre Bedingung in `if` und ggf. `otherwise`. Wenn Sie unerwartet `null` erhalten, kommt das entweder von einem fehlenden Argument (optional)

oder von einem `try`-Versuch, der `otherwise null` ergibt. Fügen Sie für `optional`-Parameter sinnvolle Default-Logik ein.
- **Rekursion zu tief:** Bei sehr hohen Werten oder unkontrollierter Rekursion kommt es zu „Stack Overflow“. Überlegen Sie, ob Sie die Funktion mit `List.Generate` oder anderen Mitteln ersetzen können. Verwenden Sie das `@` bei Selbstausruf korrekt (nur vor Funktionsname).
- **Leistungseinbruch:** Wenn der Query-Editor extrem langsam lädt, überprüfen Sie die letzten Schritte. Schlechte Kombinationen sind z.B. Sortiere in einem Schritt und dann in jeder Zeile eine komplexe Funktion. Nutzen Sie **Optimierungstipps** aus Abschnitt 2.4. Ein häufiger Fehler ist auch, Datenquellen nicht als Tabelle zu definieren, so dass Power Query immer „ganze Tabelle auslesen“ muss. Formen Sie stattdessen strukturierte Tabellen oder Abfragen.

8. Ressourcen und weiterführende Links

- **Offizielle Microsoft-Dokumentation:**
- *„Understanding Power Query M Functions“* – Grundlagen zu M-Funktionen [1] [2].
- *„Using custom functions in Power Query“* – Schritt-für-Schritt-Anleitung mit Screenshots [4] [29].
- *Fehlerbehandlung in Power Query* – Hinweise zur Nutzung von `try`, `otherwise` und `catch` [7].
- **Blogs/Tutorials:**
- The PowerUser (Miguel Escobar) – Artikel zu **rekursiven Funktionen** und anderen fortgeschrittenen Themen [14] [13].
- RADACAD (Reza Rad) – Erklärung des `#shared`-Keywords und Funktionendokumentation [27] [19].
- SSBI-Blog – Deutsche Erläuterungen zu benutzerdefinierten Funktionen und Beispiele [30] [31].
- **Community-Foren:**
- Microsoft Fabric/Power BI Community – Lösungen und Diskussionen zu gemeinsamen Funktionen und Dataflows (z.B. „Wiederverwendung einer Funktion“ [18]).
- StackOverflow und Co. – viele Q&As zu spezifischen Problemen (z.B. Umgang mit Dateien, Fehlerquellen, Leistung).
- **Bücher & Kurse:**
- *Power Query M* im Buch oder Online-Kurs (z.B. „Power Query: Dein umfassender Einstieg“ von Tobias Würffel).
- Offizielle Kurse zu Power BI und Power Query bei Microsoft Learn oder Drittanbietern.

Diese Übersicht soll Ihnen den Einstieg in benutzerdefinierte M-Funktionen erleichtern und gleichzeitig einen Pfad zu weitergehenden Techniken bieten. Die bereitgestellten Beispiele sind getestet und stehen als Vorlage für eigene Anpassungen bereit. Viel Erfolg beim Erforschen und Automatisieren mit Power Query M!

---

1 2 6 13 Understanding Power Query M functions - PowerQuery M | Microsoft Learn
https://learn.microsoft.com/en-us/powerquery-m/understanding-power-query-m-functions

3 M Language Functions - PowerQuery M | Microsoft Learn
https://learn.microsoft.com/en-us/powerquery-m/m-spec-functions

4 29 Using custom functions in Power Query - Power Query | Microsoft Learn
https://learn.microsoft.com/en-us/power-query/custom-function

5 12 28 30 31 Wie Du benutzerdefinierte Funktionen in Power Query schreibst | THE SELF-SERVICE-BI BLOG
https://ssbi-blog.de/blog/business-topics/wie-du-benutzerdefinierte-funktionen-in-power-query-schreibst/

7 Error handling - Power Query | Microsoft Learn
https://learn.microsoft.com/en-us/power-query/error-handling

8 9 List.Generate - PowerQuery M | Microsoft Learn
https://learn.microsoft.com/en-us/powerquery-m/list-generate

10 11 List.Accumulate - PowerQuery M | Microsoft Learn
https://learn.microsoft.com/en-us/powerquery-m/list-accumulate

14 Recursive Functions in Power BI / Power Query — The Power User
https://www.thepoweruser.com/2019/07/01/recursive-functions-in-power-bi-power-query/

15 Power Query Performance with Custom Functions appl... - Microsoft Fabric Community
https://community.fabric.microsoft.com/t5/Power-Query/Power-Query-Performance-with-Custom-Functions-applied-to-Excel/td-p/1637093

16 Best practices when working with Power Query - Power Query | Microsoft Learn
https://learn.microsoft.com/en-us/power-query/best-practices

17 18 25 Solved: M Power Query: how can I reuse a function between ... - Microsoft Fabric Community
https://community.fabric.microsoft.com/t5/Desktop/M-Power-Query-how-can-I-reuse-a-function-between-multiple/m-p/659775

19 20 Adding function documentation - Power Query | Microsoft Learn
https://learn.microsoft.com/en-us/power-query/handling-documentation

21 Handling unit testing for Power Query connectors - Learn Microsoft
https://learn.microsoft.com/en-us/power-query/handling-unit-testing

22 23 24 26 Power Query M Version Control using GitHub
https://www.tonymcgovern.com/blog/power-query-version-control/

27 Power Query Library of Functions; Shared Keyword - RADACAD
https://radacad.com/power-query-library-of-functions-shared-keyword/