

## Excel-Automatisierung

### Inhalt

Office Scripts in Excel – Grundlagen, Funktionsweise und Vergleich mit VBA-Makros .....	2
Anwendernamen in eine Excel-Zelle ausgeben .....	15
Automatisch Rechnungsnummern mit Excel erstellen .....	18
Kontrollkästchen als interaktives Steuerelement in Excel nutzen .....	30
So haben Sie immer alle Excel-Makros sofort zur Verfügung .....	36
Blattschutz mit Excel-Makro aktivieren .....	49

# Office Scripts in Excel – Grundlagen, Funktionsweise und Vergleich mit VBA-Makros

Office Scripts bieten einfache Aufzeichnung, Wiederverwendbarkeit, Cloud-Integration und Workflow-Fähigkeit. VBA bleibt weiterhin leistungsfähig für komplexe Szenarien. Die Entscheidung hängt vom Anwendungsfall ab. Erfahren Sie an einem Beispiel, wie Office Scripts erstellt werden und was sie von VBA unterscheidet.

Zuletzt geändert am 18.03.2026



Mit regelmäßiger Nutzung von Excel steigt der Bedarf an Automatisierung. Viele Arbeitsschritte wiederholen sich und folgen einem festen Muster. Dann kann sich der Einsatz von Automatisierungstechnologien lohnen.

Neben den klassischen **VBA-Makros** bietet Microsoft mit **Office Scripts** eine Alternative, die speziell für die Microsoft-365-Umgebung entwickelt wurde.

Dieser Beitrag erläutert:

- Was Office Scripts sind
- Wie sie funktionieren
- Welche praktischen Einsatzmöglichkeiten bestehen
- Wo Gemeinsamkeiten und Unterschiede zu VBA-Makros liegen
- Wie die weitere Entwicklung einzuordnen ist

## Automatisierung in Excel – Ausgangspunkt

Viele typische **Excel-Aufgaben** sind **wiederkehrend**:

- Daten aus ERP- oder CRM-Systemen importieren
- Spalten strukturieren

- Formatierungen anpassen
- Daten bereinigen
- Standardberichte erstellen
- Kennzahlen vorbereiten

Solche Prozesse bestehen oft aus mehreren Einzelschritten. Selbst wenn jeder Schritt nur wenige Sekunden dauert, summiert sich der **Zeitaufwand** erheblich.

Neben der Zeitersparnis spielt ein weiterer Aspekt eine große Rolle:

**Standardisierung.** Automatisierte Abläufe sorgen für:

- Einheitliche Ergebnisse
- Reduzierte Fehlerquote
- Reproduzierbarkeit
- Dokumentierbarkeit von Prozessschritten

Excel bietet hierfür zwei Technologien:

- VBA-Makros
- Office Scripts

Während VBA seit den 1990er-Jahren existiert, sind Office Scripts eine neue Entwicklung.

## Was sind Office Scripts?

Office Scripts sind Skripte zur **Automatisierung von Aufgaben in Excel** innerhalb von **Microsoft 365**. Sie ermöglichen es, definierte Arbeitsschritte automatisch auszuführen. Dabei können Scripts:

- manuell per Klick gestartet werden,
- in Workflows eingebunden werden und
- mehrfach in unterschiedlichen Dateien verwendet werden.

## Technologische Grundlage der Office Scripts

Office Scripts basieren auf **TypeScript**, einer Weiterentwicklung von JavaScript. Im Hintergrund greifen Office Scripts auf eine Programmierschnittstelle (ExcelScript-API) zu. Diese API stellt Funktionen bereit, um:

- Arbeitsblätter zu steuern
- Zellbereiche auszuwählen
- Inhalte zu lesen oder zu verändern
- Formatierungen anzuwenden
- Tabellenobjekte zu erstellen

Für Anwender ist wichtig: Die **Aufzeichnungsfunktion** erzeugt den notwendigen Code automatisch.

## Verfügbarkeit und Systemumgebung

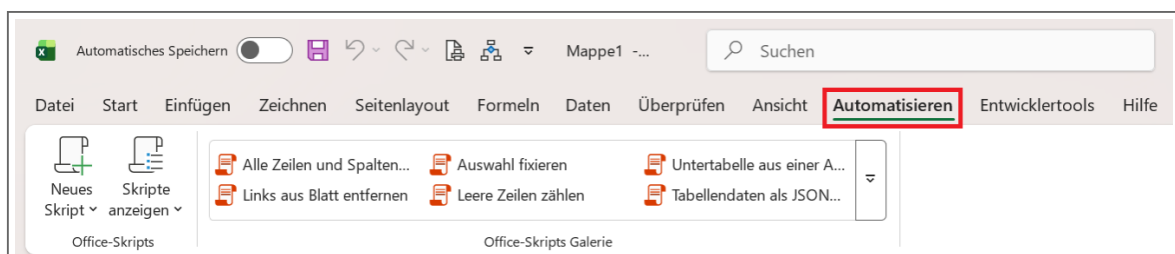
Office Scripts sind verfügbar in:

- Excel für das Web
- Excel für Windows (Microsoft-365-Abonnement)

Sie sind nicht verfügbar in:

- Excel 2016 oder 2019 als Dauerlizenz
- älteren On-Premise-Installationen

Die Funktion befindet sich in der Registerkarte **Automatisieren**.



*Excel-Menü: Automatisieren mit Office Scripts*

Ein wesentlicher Unterschied zu VBA liegt im **Speicherort**:

- VBA-Makros sind Bestandteil der Datei (.xlsm).
- Office Scripts werden im Benutzerkonto gespeichert.

Dadurch können **Office Scripts unabhängig von einer einzelnen Datei** verwendet werden.

## Wie funktionieren Office Scripts technisch?

Ein Office Script folgt einer klar strukturierten **Logik**. Ein typisches Script:

1. Greift auf die aktuelle Arbeitsmappe zu.
2. Wählt ein Arbeitsblatt.
3. Bestimmt einen Zellbereich.
4. Führt Aktionen aus.

Beispiel:

```
function main(workbook: ExcelScript.Workbook) {  
    let sheet = workbook.getActiveWorksheet();  
    let range = sheet.getUsedRange();  
    range.getFormat().getFont().setBold(true);  
}
```

Die Struktur ist systematisch aufgebaut:

- **workbook** repräsentiert die Arbeitsmappe
- **getActiveWorksheet()** greift auf das aktuelle Blatt zu
- **getUsedRange()** ermittelt den genutzten Bereich
- **getFormat()** als Formatierungsfunktion führt die Aktion aus

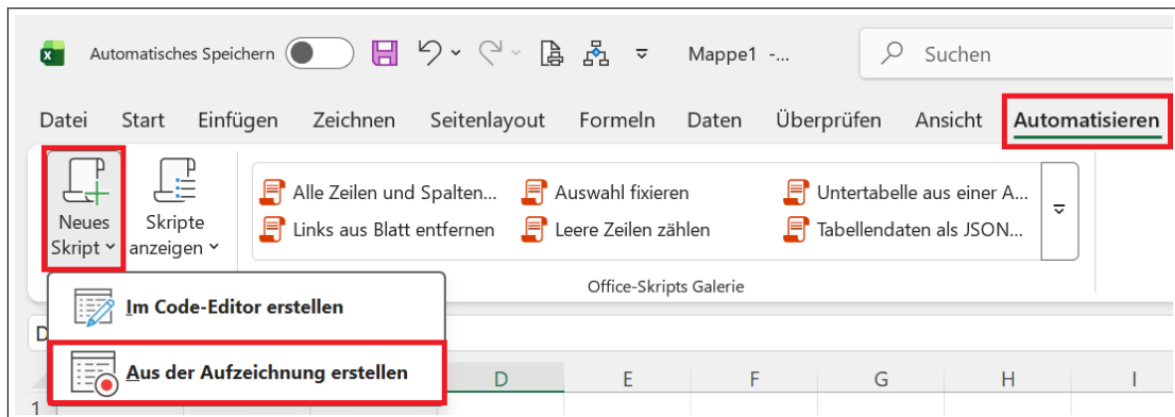
Das Objektmodell ähnelt in seiner Logik dem VBA-Objektmodell, ist jedoch moderner aufgebaut.

## Erste Schritte mit Office Scripts

Das folgende Beispiel zeigt den Einsatz von Office Scripts an einem Beispiel. Es soll ein Skript erstellt werden, mit dem man den Blattschutz in einem Tabellenblatt aktivieren kann.

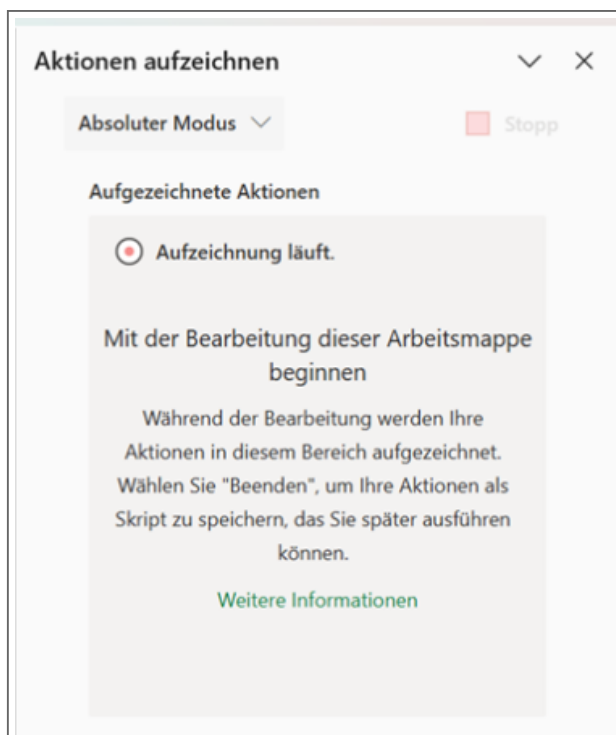
## 1. Script aufzeichnen

Aktivieren Sie im Menüband die Befehlsfolge Registerkarte **Automatisieren** > Befehl **Neues Skript** > **Aus der Aufzeichnung erstellen**.



Menü-Funktion Neues Skript aufzeichnen

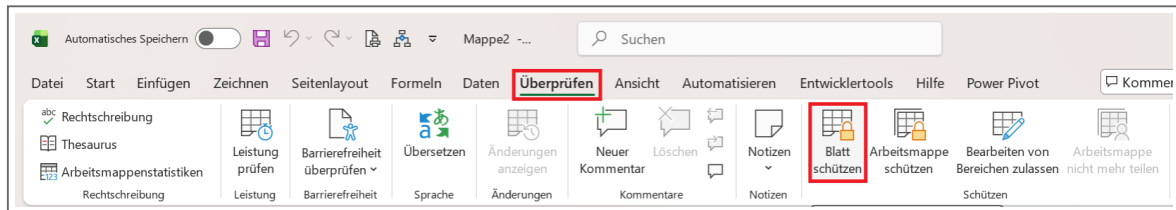
Am rechten Rand des Fensters öffnet sich der Aufgabenbereich **Aktionen aufzeichnen**. Hier wird der aktuelle Status zu der Aufzeichnung dargestellt.



Protokoll der Skript-Aufzeichnung in Excel

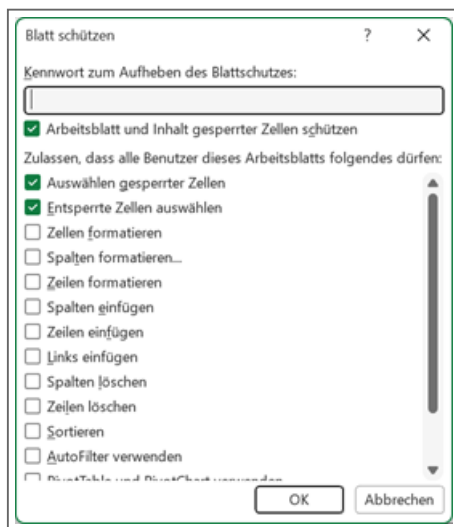
Zeichnen Sie nun die gewünschten Bearbeitungsschritte auf. Im Beispiel soll das Tabellenblatt geschützt werden.

Aktivieren Sie dazu im Menüband die Befehlsfolge Registerkarte **Überprüfen** > Befehlsgruppe **Schützen** > Befehl **Blatt schützen**.



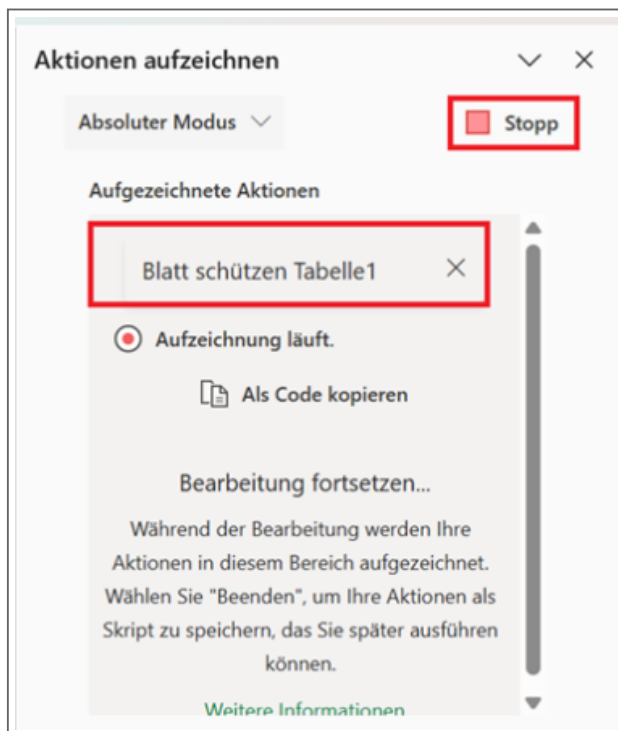
*Beispiel: Befehlsabfolge Blatt schützen*

Es öffnet sich das Dialogfeld **Blatt schützen**. Bestätigen Sie hier die Voreinstellung und klicken Sie auf die Schaltfläche **OK**.

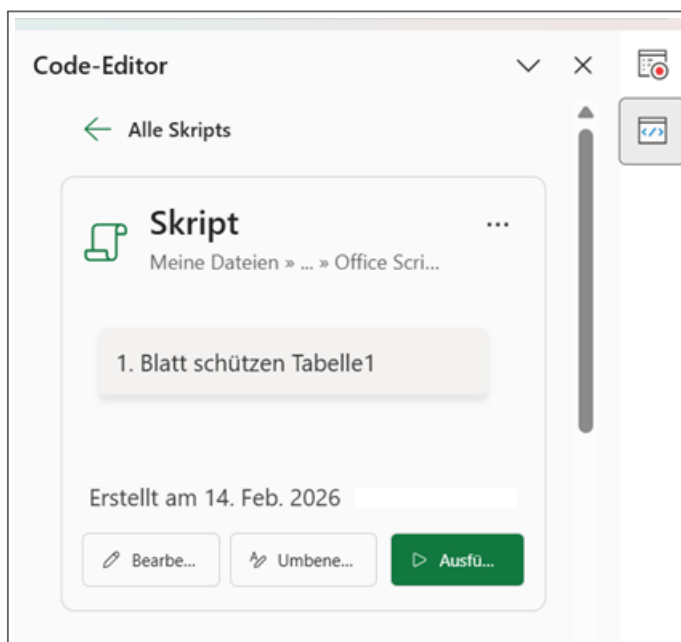


*Blattschutz in Excel aktivieren*

Der Bearbeitungsschritt wird automatisch im Aufgabenbereich am rechten Rand dargestellt. Stoppen Sie jetzt die Skript-Aufzeichnung, indem Sie im **Aufgabenbereich** auf **Stopp** klicken.

*Skript-Aufzeichnung beenden*

Sie bekommen dann das aufgezeichnete Skript im Aufgabenbereich angezeigt.

*Ergebnis der Skript-Aufzeichnung*



Excel generiert automatisch ein Skript. Über die Schaltflächen am unteren Ende können Sie den Code bearbeiten oder anzeigen lassen (**Bearbeiten**), den Skriptnamen ändern (**Umbenennen**) oder das Skript ausführen lassen (**Ausführen**).

## Typische Anwendungsfälle

Office Scripts eignen sich besonders für:

- Standardisierung von Tabellenlayouts
- Automatisches Erstellen von Tabellen
- Entfernen leerer Zeilen
- Formatierung von Exportdateien
- Vorbereitung von Monatsreports

Der Zeitgewinn kann je nach Umfang erheblich sein.

## Unterschiede zwischen VBA-Makros und Office Scripts

Auch wenn Office Scripts und VBA-Makros dasselbe Ziel verfolgen – nämlich die Automatisierung von Excel – unterscheiden sie sich in ihrer Architektur, technischen Grundlage und strategischen Ausrichtung grundlegend.

Ein genauer Blick hilft dabei, die jeweiligen Stärken besser einzuordnen.

Architektur – lokal und dateibasiert vs. cloudorientiert und kontobasiert

**VBA (Visual Basic for Applications)** ist tief in die Desktop-Version von Excel integriert, weil lokal und dateigebunden. Makros werden direkt in der jeweiligen Arbeitsmappe gespeichert. Damit sind sie:

- Bestandteil der Datei,
- an das Dateiformat (.xlsm oder .xlsb) gebunden und
- lokal ausführbar.

Das bedeutet: Öffnen Sie eine Datei mit Makros, ist der VBA-Code Bestandteil dieser Datei. Die Automatisierungslogik reist sozusagen mit dem Dokument mit.

Diese Architektur hat Vorteile:

- Makros sind direkt mit der Datei verknüpft.
- Sie können sehr individuell auf diese Datei zugeschnitten sein.
- Sie funktionieren vollständig offline.

Gleichzeitig entstehen aber auch Einschränkungen:

- Jede Datei enthält ihre eigene Makrologik.
- Versionierung und Wartung können aufwendig werden.
- Verteilung aktualisierter Makros ist organisatorisch anspruchsvoll.

**Office Scripts** verfolgen einen anderen Ansatz. Sie sind cloudorientiert und kontobasiert. Sie werden nicht in der Excel-Datei gespeichert, sondern im Microsoft-Konto des Anwenders. Das hat mehrere Konsequenzen:

- Ein Skript ist nicht an eine bestimmte Datei gebunden.
- Es kann in unterschiedlichen Arbeitsmappen verwendet werden.
- Die Datei selbst bleibt eine normale .xlsx-Datei.

Architektonisch betrachtet sind Office Scripts stärker serviceorientiert aufgebaut. Sie greifen über eine moderne Programmierschnittstelle (API) auf Excel zu. Das passt zur Microsoft-365-Strategie:

- Dateien liegen häufig in OneDrive oder SharePoint.
- Mehrere Personen arbeiten parallel.
- Prozesse werden zunehmend zentral verwaltet.

Für Anwender bedeutet das: Office Scripts eignen sich besonders gut für standardisierte Abläufe, die in mehreren Dateien identisch ausgeführt werden sollen.

## Programmiersprache – VBA vs. TypeScript

**VBA** ist eine proprietäre und klassische Office-Automatisierungssprache, die speziell für die Automatisierung von Microsoft-Office-Anwendungen entwickelt wurde.

Eigenschaften von VBA:

- Seit den 1990er-Jahren etabliert
- eng an das Office-Objektmodell gekoppelt
- sehr mächtig innerhalb der Desktop-Welt
- imperativ aufgebaut

VBA ist tief in Excel integriert. Das ermöglicht direkte Steuerungsmöglichkeiten, etwa:

- Zugriff auf Dateisysteme
- Interaktion mit Windows
- Steuerung anderer Office-Anwendungen

Allerdings basiert VBA auf einer älteren Technologiearchitektur. Es ist nicht webbasiert und nicht für moderne Cloud-Umgebungen konzipiert.

**Office Scripts** nutzen TypeScript – eine Weiterentwicklung von JavaScript. TypeScript ist:

- modern,
- objektorientiert,
- weitverbreitet in der Webentwicklung und
- stark typisiert.

Der Vorteil dieser Technologie:

- bessere Strukturierbarkeit
- klar definierte Schnittstellen
- moderne Entwicklungsumgebung
- gute Erweiterbarkeit

Für Anwender ist die konkrete Sprache oft weniger entscheidend. Relevant wird sie dann, wenn Code angepasst oder erweitert werden soll. Langfristig ist erkennbar, dass Microsoft auf webbasierte Technologien setzt. Office Scripts fügen sich nahtlos in diese Strategie ein.

## Sicherheit – Makrowarnungen vs. Microsoft-365-Integration

**VBA-Makros** gelten aus Sicherheitsgründen als potenzielles Risiko. Beim Öffnen einer Datei mit Makros erscheint häufig eine Warnmeldung: „Makros wurden deaktiviert.“

Der Hintergrund: VBA-Code kann sehr weitreichende Aktionen ausführen – inklusive Zugriff auf lokale Dateien oder Systemressourcen. Das macht Makros leistungsfähig, aber auch anfällig für Missbrauch.

Unternehmen setzen daher oft restriktive Richtlinien ein:

- Makros werden standardmäßig deaktiviert.
- Nur signierte Makros sind erlaubt.
- Die Ausführung ist eingeschränkt.

**Office Scripts** sind in die Sicherheitsarchitektur von Microsoft 365 eingebettet. Das bedeutet:

- Zugriff erfolgt über definierte APIs.
- Kein direkter Zugriff auf lokale Systemressourcen.
- Ausführung innerhalb einer kontrollierten Umgebung.

Da Office Scripts primär auf Excel-Objekte zugreifen und nicht auf das Betriebssystem, ist das Risikoprofil anders gelagert. Das heißt nicht, dass keinerlei Sicherheitsaspekte bestehen – aber die Architektur ist stärker kontrolliert und servicebasiert.

## Funktionsumfang – tiefe Systemintegration vs. Excel-Fokus

**VBA** bietet umfangreiche Möglichkeiten:

- Benutzerformulare (UserForms)
- Dialogfenster
- Zugriff auf lokale Dateien
- Steuerung anderer Office-Anwendungen
- Zugriff auf Windows-APIs
- Komplexe Ereignissteuerung

Damit lassen sich sehr individuelle Lösungen entwickeln – bis hin zu vollständigen Mini-Anwendungen innerhalb von Excel. Gerade bei historisch gewachsenen Lösungen ist VBA oft tief in Prozesse integriert.

**Office Scripts** konzentrieren sich primär auf Excel-interne Prozesse wie:

- Manipulation von Zellbereichen
- Tabellen

- Formatierungen
- Datenverarbeitung
- Strukturierung von Arbeitsmappen

Nicht vorgesehen sind derzeit:

- Klassische Benutzerformulare
- Direkter Zugriff auf lokale Dateien
- Tiefe Systeminteraktion

Das macht Office Scripts weniger flexibel im systemnahen Bereich, aber klarer fokussiert auf standardisierte Excel-Prozesse.

## Integration in Workflows – lokale Automatisierung vs. Prozessintegration

Ein besonders relevanter Unterschied liegt in der Integration in übergeordnete Automatisierungsprozesse.

**VBA** ist traditionell auf lokale Automatisierung ausgelegt. Makros werden:

- beim Öffnen einer Datei,
- per Button oder
- über Ereignisse in Excel ausgeführt.

Zwar lassen sich auch mit VBA komplexe Automatisierungen erstellen, doch die Einbindung in cloudbasierte Workflow-Systeme ist nicht der ursprüngliche Fokus.

**Office Scripts** lassen sich direkt in Power Automate einbinden. Das ermöglicht Szenarien wie:

- Eine Datei wird in einen SharePoint-Ordner hochgeladen → Script läuft automatisch.
- Jeden Morgen wird ein Report erzeugt und per E-Mail versendet.
- Daten werden aus einer Excel-Datei verarbeitet und in ein anderes System übertragen.

Hier wird Excel Teil eines größeren, automatisierten Workflows. Office Scripts fungieren dabei als ausführendes Element innerhalb einer Prozesskette.

## Zusammenfassung der Unterschiede VBA und Office Scripts

VBA und Office Scripts unterscheiden sich nicht nur technisch, sondern auch strategisch:

- VBA ist tief integriert, systemnah und dateibasiert
- Office Scripts sind API-basiert, cloud-orientiert und workflowfähig

Für Anwender bedeutet das:

- VBA bietet maximale Flexibilität im Desktop-Bereich
- Office Scripts bieten moderne Integration und Standardisierung

Beide Technologien haben ihre Berechtigung – jedoch in unterschiedlichen Kontexten.

## Weitere Entwicklung von Office Scripts

Die Entwicklung von Office Scripts ist Teil einer größeren Transformation der Office-Produkte. Microsoft setzt zunehmend auf:

- Cloud-Integration
- Plattformunabhängigkeit
- API-basierte Erweiterbarkeit
- Vernetzung von Diensten

Office Scripts sind darauf ausgelegt, Excel in moderne Prozessketten einzubinden. Das bedeutet jedoch nicht das Ende von VBA. VBA bleibt relevant für:

- bestehende komplexe Lösungen
- Desktop-nahe Automatisierungen
- Spezialanwendungen

Langfristig ist jedoch erkennbar: Neue Automatisierungsfunktionen entstehen vor allem im Microsoft-365-Umfeld. Für Anwender ergibt sich eine realistische Perspektive:

- Bestehende Makros bleiben nutzbar.
- Neue Automatisierungen können mit Office Scripts aufgebaut werden.
- Beide Technologien ergänzen sich.

## Anwendernamen in eine Excel-Zelle ausgeben

Der Name eines Anwenders, der eine Excel-Datei gerade bearbeitet, lässt sich mit einer Funktion ermitteln und in der Tabelle ausgeben. Wie gehen Sie dazu vor?

Zuletzt geändert am 18.03.2026

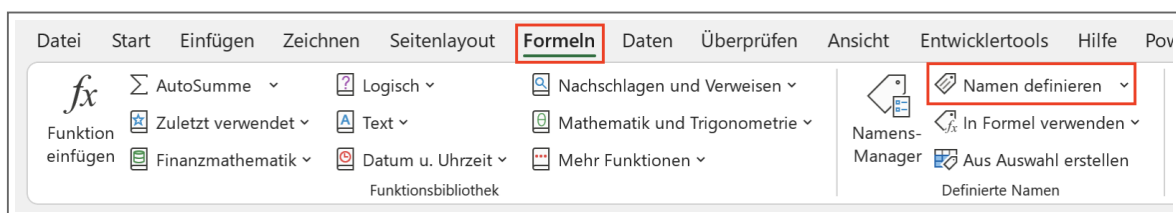


Manche Excel-Dateien werden von unterschiedlichen Anwendern im Unternehmen genutzt und bearbeitet. Je nachdem, wer die Datei gerade bearbeitet, sollen bestimmte Zellen sichtbar, versteckt oder zu bearbeiten sein.

Dazu müssen Sie in der Datei ermitteln, wer sie gerade bearbeitet.

Wollen Sie den Anwendernamen des aktuellen Users in eine Zelle ausgeben? Hierfür können Sie eine alte Excel4-Makrofunktion einsetzen.

Definieren Sie zuerst einen Namen in der entsprechenden Arbeitsmappe, indem Sie im Menüband die Befehlsfolge Registerkarte **Formeln** > Befehlsgruppe **Definierte Namen** > Befehl **Namen definieren** aktivieren.

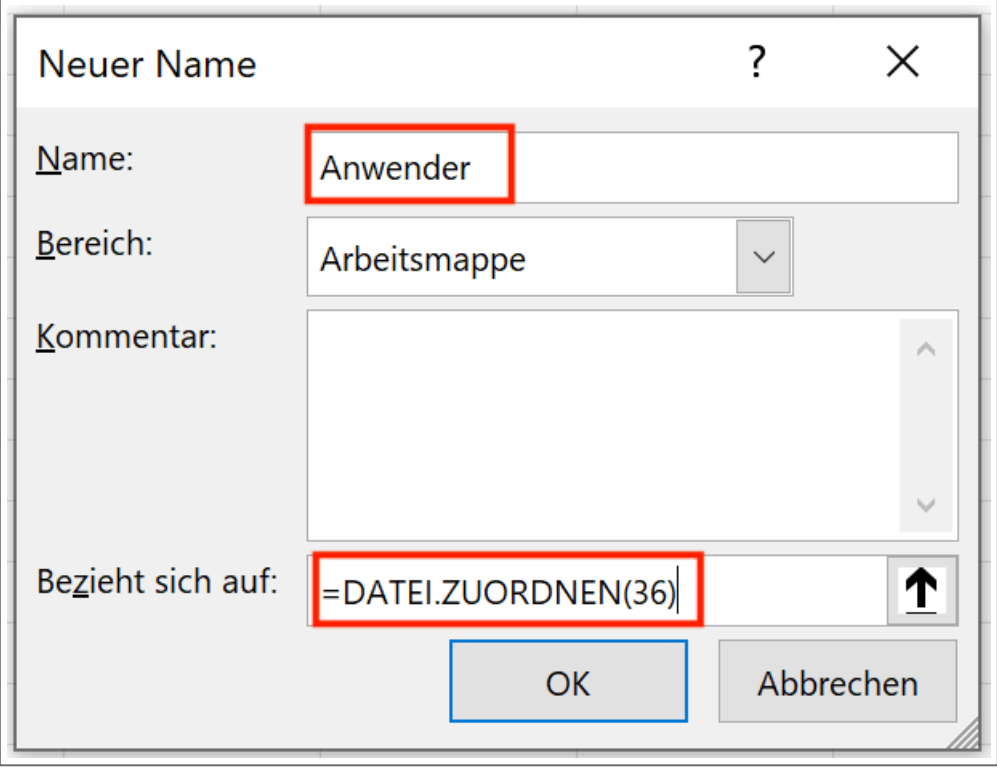


*Namen definieren in Excel*

Daraufhin öffnet sich das Dialogfeld **Neuer Name**. Vergeben Sie hier unter **Name:** einen aussagekräftigen Namen, zum Beispiel **Anwender**.

Unter **Bezieht sich auf:** erfassen Sie jetzt die Excel4-Makrofunktion **=DATEI.ZUORDNEN(36)**.

Bestätigen Sie den definierten Namen, indem Sie auf die Schaltfläche **OK** klicken.



The screenshot shows the 'Neuer Name' (New Name) dialog box in Microsoft Excel. The dialog has a title bar with a question mark and a close button. It contains four main fields: 'Name:' with the value 'Anwender', 'Bereich:' (Scope) with a dropdown menu showing 'Arbeitsmappe', 'Kommentar:' (Comment) with an empty text area, and 'Bezieht sich auf:' (Refers to) with the formula '=DATEI.ZUORDNEN(36)'. The 'Name' and 'Bezieht sich auf' fields are highlighted with red rectangles. At the bottom, there are two buttons: 'OK' (highlighted with a blue border) and 'Abbrechen' (Cancel).

*Anwendername dem Formelnamen zuordnen*

Jetzt können Sie sich den Namen des aktuellen Users anzeigen lassen, der die Arbeitsmappe geöffnet hat, indem Sie mit einer Formel auf den definierten Namen verweisen: **=Anwender**

Als Ergebnis erhalten Sie in der Zelle den Namen des aktuellen Anwenders angezeigt.



A1				
✕ ✓ <i>fx</i> =Anwender				
	A	B	C	D
1	Jürgen Fleig			
2				
3				
4				
5				
6				
7				
8				

Ausgabe: Name des aktuellen Anwenders der Excel-Datei

# Automatisch Rechnungsnummern mit Excel erstellen

Mit zwei unterschiedlichen Methoden können Sie schnell und zuverlässig eine große Anzahl an Rechnungsnummern in Excel erzeugen – entweder mit einer einfachen Formel oder mit Power Query für maximale Automatisierung.

Zuletzt geändert am 18.03.2026



## Die Aufgabe: Nummern systematisch festlegen

In vielen Unternehmen müssen regelmäßig Rechnungs- oder andere **Nummern** vergeben werden, die einem bestimmten **Schema** folgen. Eine manuelle Erstellung ist fehleranfällig und zeitraubend.

Excel bietet jedoch eine einfache Möglichkeit, diese Nummern automatisch zu generieren. In diesem Beitrag lernen Sie zwei Methoden kennen, um eine Liste von fortlaufenden Rechnungsnummern zu erstellen: mit einer Excel-Formel und mit Power Query.

## Beispiel: Aufbau einer Rechnungsnummer

Die Rechnungsnummer für das folgende Beispiel soll aus drei Bestandteilen bestehen:

- Präfix: Text "KD"
- Nummer: Fortlaufende Zahl von 1 bis 1000 (8-stellig mit führenden Nullen)
- Suffix: "-2025"

Das Ergebnis soll also wie folgt aussehen:

- KD00000001-2025
- KD00000002-2025 ...
- KD00001000-2025

**Hinweis:** Mit den folgenden beiden Methoden können Sie beliebige andere Nummern- oder Zeichenfolgen als lange Listen erzeugen; je nachdem, wofür Sie diese benötigen.

## Methode 1: Automatische Erstellung mit einer Excel-Formel

Mit einer einzigen Excel-Formel können Sie die gesamte Liste generieren, ohne dass manuell Werte eingetragen oder gezogen werden müssen. Verwenden Sie dazu die folgende Formel in einer beliebigen Zelle:

**= "KD" & TEXT(SEQUENZ(1000;1;1;1);"00000000") & "-2025"**

Die Formel funktioniert folgendermaßen:

- **SEQUENZ(1000;1;1;1)** erzeugt eine Liste von 1000 Zahlen, beginnend bei 1, mit einer Schrittweite von 1.
- **TEXT(...;"00000000")** formatiert die Zahlen so, dass sie stets 8-stellig sind, indem führende Nullen ergänzt werden.
- **"KD" & ...** setzt das Präfix „KD“ vor die formatierte Zahl.
- **... & "-2025"** fügt das Suffix „-2025“ an das Ende an.

Diese Methode besitzt folgende Vorteile:

- Kein manuelles Ziehen notwendig – die Liste wird sofort erstellt.
- Automatische Formatierung – die Nummer bleibt immer 8-stellig.
- Flexible Anpassung – Anzahl der Rechnungsnummern, Präfix, Suffix und das Format können leicht verändert werden.

Das Ergebnis ist ein Array in der Spalte unter der eingegebenen Funktion. Dort benötigen Sie entsprechend freie Zellen, damit keine Fehlermeldung erscheint.

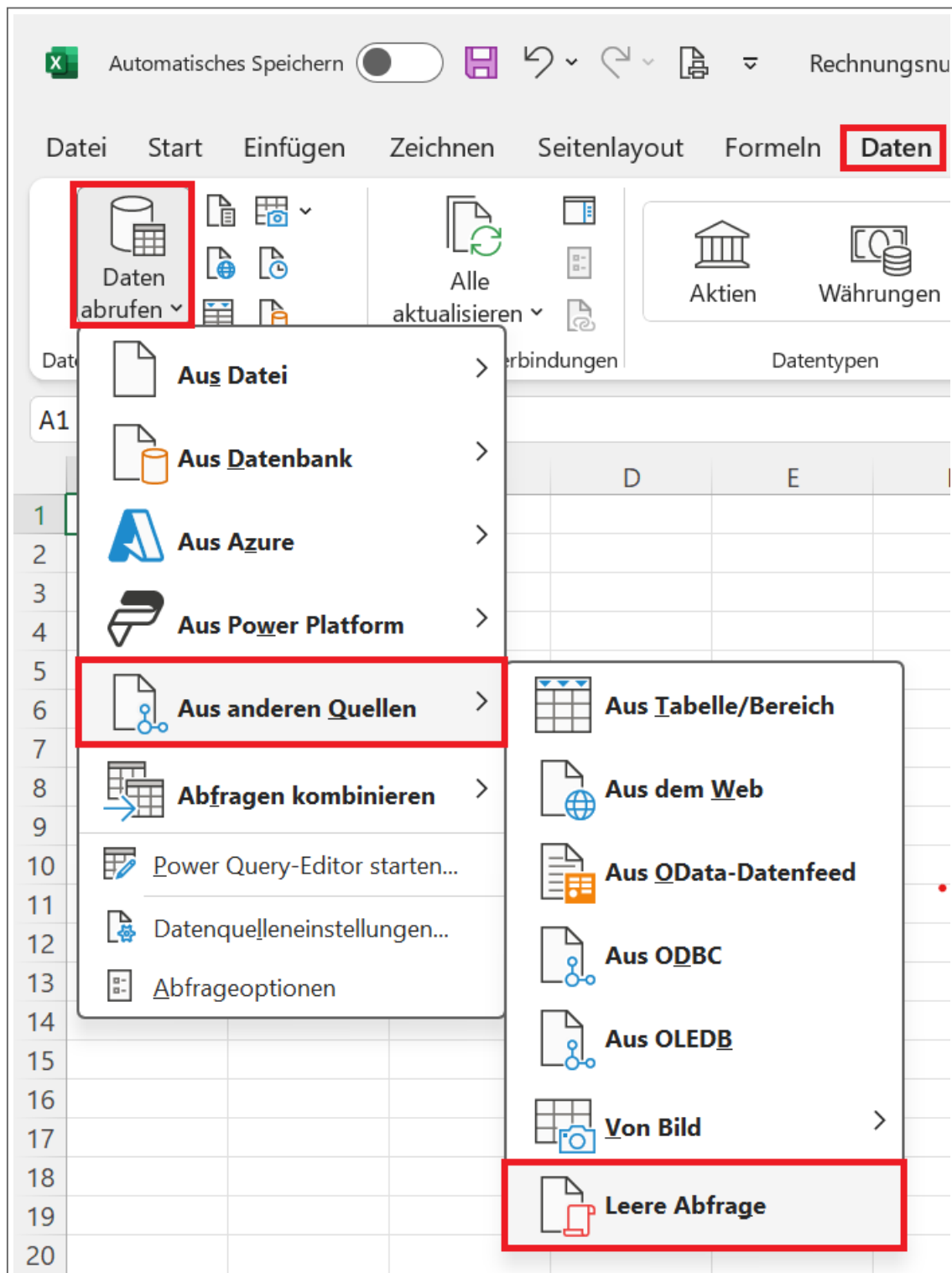
### Tipp

Falls Sie die Rechnungsnummern als feste Werte speichern möchten, kopieren Sie die generierte Spalte und fügen Sie diese mit der Option **Werte einfügen** an einer anderen Stelle Ihrer Tabelle ein.

## Methode 2: Automatische Erstellung der Nummernfolge mit Power Query

Eine weitere leistungsstarke Möglichkeit ist die Verwendung von Power Query, um die Rechnungsnummern zu generieren.

Öffnen Sie eine leere Abfrage in Power Query, indem Sie im Menüband die folgende Befehlsfolge ausführen: Registerkarte **Daten** > Befehlsgruppe **Daten abrufen und transformieren** > Befehl **Daten abrufen** > Befehl **Aus anderen Quellen** > **Leere Abfrage**.

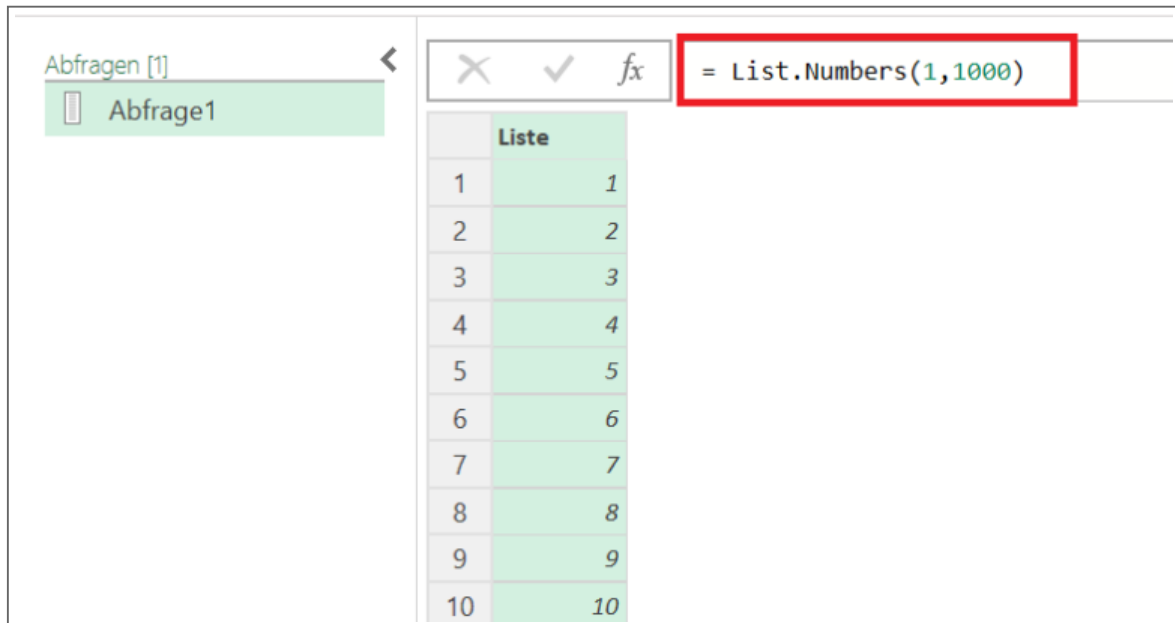


Leere Power-Query-Abfrage erzeugen

Es öffnet sich eine leere Abfrage. Erfassen Sie in der Bearbeitungsleiste die folgende Formel und bestätigen Sie anschließend mit Enter:

**=List.Numbers(1,1000)**

Es wird nun eine Liste von 1 bis 1000 erstellt.

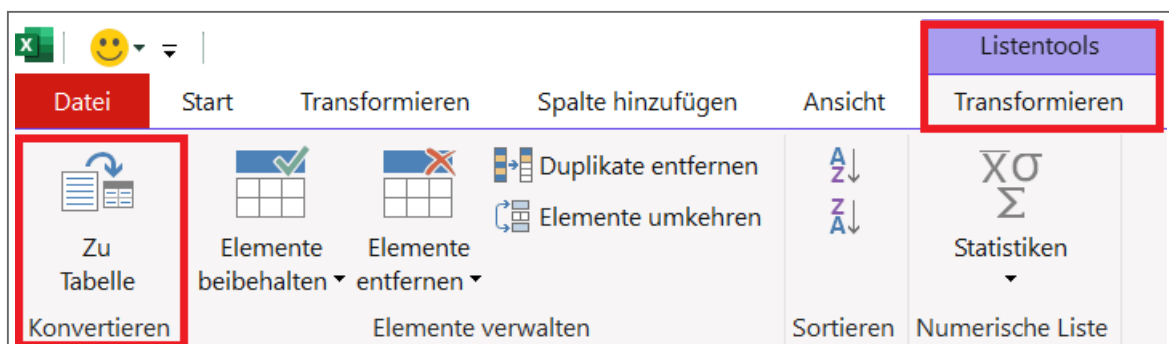


The screenshot shows the Power Query Editor interface. On the left, the 'Abfragen [1]' pane contains 'Abfrage1'. The formula bar at the top right displays the formula `= List.Numbers(1,1000)`, which is highlighted with a red rectangle. Below the formula bar, a preview table titled 'Liste' is shown, containing a column of numbers from 1 to 10.

	Liste
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Liste der Zahlen 1 bis 1000 in Power Query erzeugen

Wandeln Sie die Liste nun in eine Tabelle um, indem Sie im Menüband den Befehl **Zu Tabelle** auswählen.



Liste in Tabelle mit Power Query umwandeln

Es öffnet sich das Dialogfeld **Zu Tabelle**. Lassen Sie hier die Einstellungen unverändert und bestätigen Sie die Einstellungen, indem Sie das Dialogfeld durch einen Klick auf **OK** schließen.

**Zu Tabelle**

Erstellen Sie eine Tabelle aus einer Liste von Werten.

Trennzeichen eingeben oder auswählen  
Keine

Behandlung zusätzlicher Spalten  
Als Fehler anzeigen

OK Abbrechen

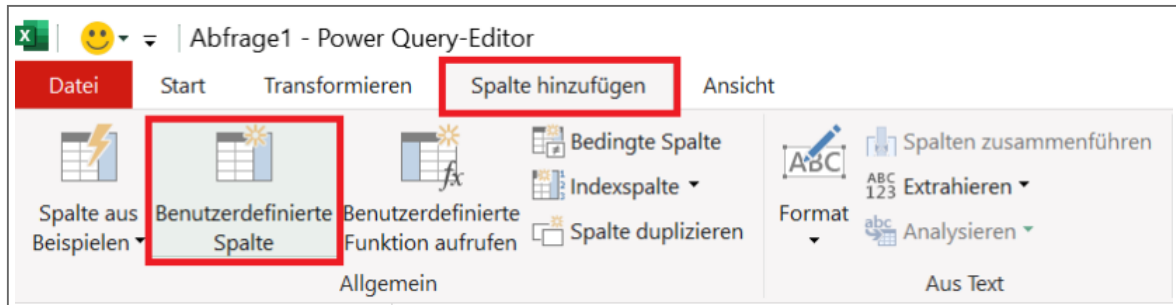
*Einstellungen für die Umwandlung einer Liste in eine Tabelle*

Die Liste wird nun in eine normale Tabelle umgewandelt. Die Liste erhält eine Spaltenüberschrift mit dem Namen Column1.

	Column1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

*Power-Query-Tabelle mit den Zahlen 1 bis 1000 (Auszug)*

Aktivieren Sie nun im Menüband die Befehlsfolge **Spalte hinzufügen** > **Benutzerdefinierte Spalte**, um eine neue Spalte einzufügen.

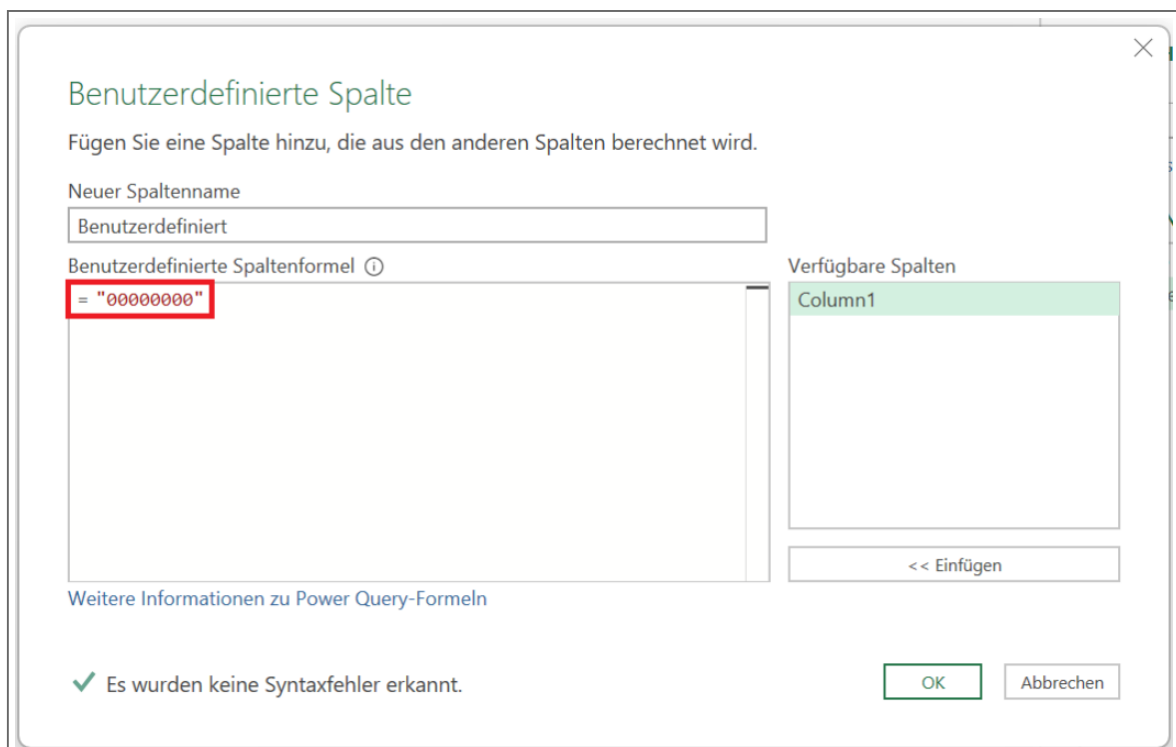


Power Query: Benutzerdefinierte Spalte hinzufügen

Es öffnet sich das Dialogfeld **Benutzerdefinierte Spalte**. Erfassen Sie unter **Benutzerdefinierte Spaltenformel** die Formel

**= "00000000"**

Mit dieser Formel erstellen Sie einen Text von 8 Nullen. Bestätigen Sie Ihre Einstellungen, indem Sie das Dialogfeld durch einen Klick auf **OK** schließen.



Spalte mit einem festgelegten Wert einfügen



Daraufhin wird eine neue Spalte mit dem Text **00000000** eingefügt.

	ABC 123 Column1	ABC 123 Benutzerdefiniert
1		1 00000000
2		2 00000000
3		3 00000000
4		4 00000000
5		5 00000000
6		6 00000000
7		7 00000000
8		8 00000000
9		9 00000000
10		10 00000000

*Spalte mit festgelegtem Wert*

Bei beiden Spalten wird bislang der Datentyp **ABC123** angezeigt. Wandeln Sie den Datentyp bei beiden Spalten in **Text (ABC)** um.

Klicken Sie dazu mit der linken Maustaste auf das Symbol (ABC123) in den Spalten und wählen Sie anschließend in der Liste den Datentyp **ABC** für Text aus.

<div> <span>✕</span> <span>✓</span> <span><i>fx</i></span> </div> <div>= Table.TransformColumnTypes("#Hinz</div>		
	<b>Column1</b>	<b>Benutzerdefiniert</b>
1	1	00000000
2	2	00000000
3	3	00000000
4	4	00000000
5	5	00000000
6	6	00000000
7	7	00000000
8	8	00000000
9	9	00000000
10	10	00000000

*Datentyp in Power Query ändern*

Fügen Sie nun eine weitere benutzerdefinierte Spalte hinzu (**Spalte hinzufügen > Benutzerdefinierte Spalte**) und erfassen Sie die folgende Formel:

**= "KD" & Text.End([Benutzerdefiniert]&[Column1],8) & "-2025"**

Mit dem Formelteil **[Benutzerdefiniert]&[Column1]** werden die beiden Spalten verkettet. Sie erhalten hier somit die folgenden Werte:

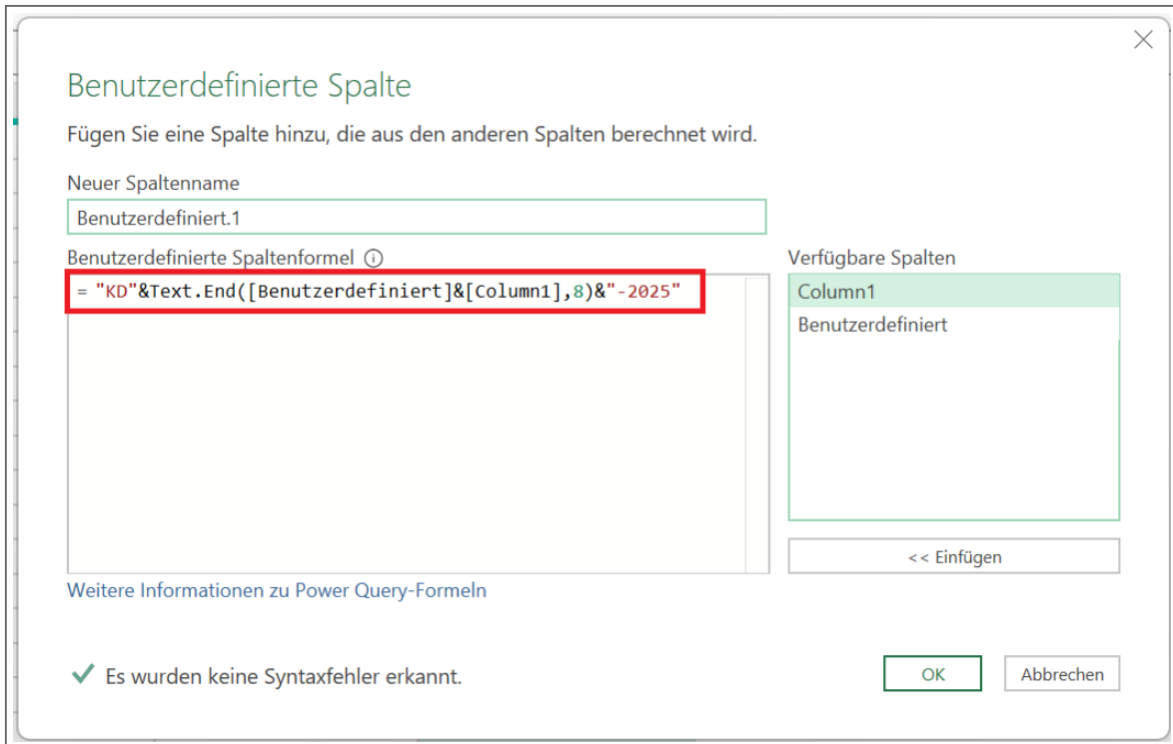
- 000000001
- 000000002 ...
- 000000001000

Mit dem Formelteil **Text.End([Benutzerdefiniert]&[Column1],8)** werden anschließend von rechts die ersten 8 Zeichen ermittelt. Sie erhalten somit die folgenden Werte:

- 00000001
- 00000002 ...
- 00001000

Die Nummern sind jetzt 8-stellig mit führenden Nullen.

Mit **= "KD"&Text.End([Benutzerdefiniert]&[Column1],8)&"-2025"** werden der Präfix (KD) und der Suffix (-2025) an die Nummer angefügt.



**Benutzerdefinierte Spalte**

Fügen Sie eine Spalte hinzu, die aus den anderen Spalten berechnet wird.

Neuer Spaltenname  
Benutzerdefiniert.1

Benutzerdefinierte Spaltenformel ⓘ  
= "KD"&Text.End([Benutzerdefiniert]&[Column1],8)&"-2025"

Verfügbare Spalten  
Column1  
Benutzerdefiniert

<< Einfügen

Weitere Informationen zu Power Query-Formeln

✓ Es wurden keine Syntaxfehler erkannt.

OK Abbrechen

Formel in Power Query für das Verknüpfen mehrerer Spalten

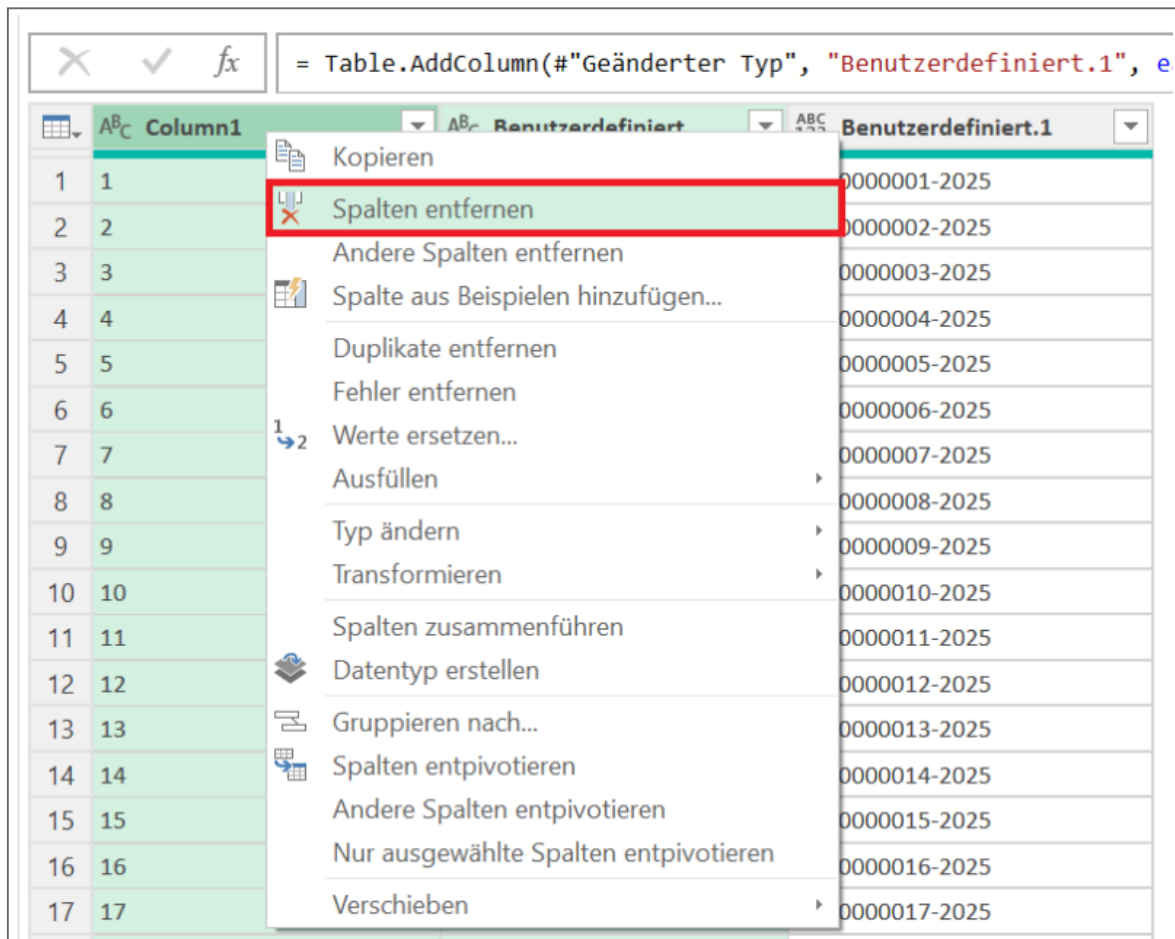
Es wird eine neue Spalte mit den gewünschten Rechnungsnummern eingefügt.

	ABC Column1	ABC Benutzerdefiniert	ABC 123 Benutzerdefiniert.1
1	1	00000000	KD00000001-2025
2	2	00000000	KD00000002-2025
3	3	00000000	KD00000003-2025
4	4	00000000	KD00000004-2025
5	5	00000000	KD00000005-2025
6	6	00000000	KD00000006-2025
7	7	00000000	KD00000007-2025
8	8	00000000	KD00000008-2025
9	9	00000000	KD00000009-2025
10	10	00000000	KD00000010-2025

Ergebnis: Neue Spalte mit den erstellten Rechnungsnummern

Entfernen Sie nun die ersten beiden Spalten, indem Sie diese markieren und anschließend mit der **rechten Maustaste** auf einem markierten Spaltentitel klicken.

Wählen Sie im Kontextmenü den Eintrag **Spalten entfernen** aus.



*Unnötige Spalten in Power Query löschen*

Die beiden Spalten werden daraufhin entfernt. Es verbleibt nur noch die Spalte mit den Rechnungsnummern.

Benennen Sie diese Spalte jetzt noch in Rechnungsnummer um, indem Sie mit der **linken Maustaste** doppelt auf den Spaltentitel klicken. Sie können daraufhin die Spaltenüberschrift umbenennen.

  $fx$		= Table.Ren
	ABC 123	Rechnungsnummer 
1	KD00000001-2025	
2	KD00000002-2025	
3	KD00000003-2025	
4	KD00000004-2025	
5	KD00000005-2025	
6	KD00000006-2025	
7	KD00000007-2025	
8	KD00000008-2025	
9	KD00000009-2025	
10	KD00000010-2025	

*Tabelle in Power Query mit den individuell erstellten Kundennummern*

Sie können diese Liste nun in die gewünschte Excel-Tabelle übertragen, indem Sie im Menüband die Befehlsfolge **Datei > Schließen & laden** anklicken.

# Kontrollkästchen als interaktives Steuerelement in Excel nutzen

Kontrollkästchen in Excel sind eine bahnbrechende Verbesserung für alle, die interaktive Arbeitsmappen erstellen möchten – sei es zur Aufgabenverwaltung, zur Filterung von Datensätzen oder zur Steuerung von Dashboards. So setzen Sie Kontrollkästchen (Checkboxes) einfach, formelbasiert und systemübergreifend ein – ohne Tricks oder Makros.

Zuletzt geändert am 18.03.2026



Interaktive **Kontrollkästchen**, die direkt in Zellen einer Excel-Tabelle eingebettet sind, lassen sich in Microsoft 365 Excel ohne VBA oder Formularsteuerelemente verwenden.

Sie bieten eine einfache, moderne Möglichkeit zur **Dateneingabe** – ganz im Stil dynamischer Tabellen, Dashboards oder To-do-Listen.

In diesem Beitrag erfahren Sie, wie Sie das Kontrollkästchen in Excel aktivieren, anpassen und produktiv einsetzen – inklusive Beispiele aus der Praxis.

## Was ist besonders am Kontrollkästchen?

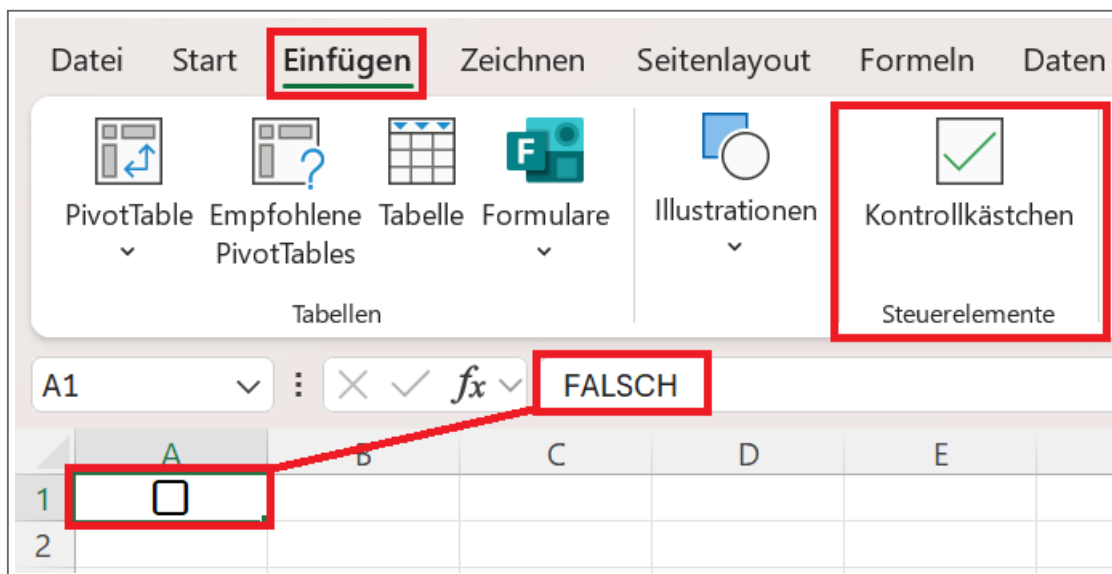
Im Gegensatz zum früheren Umweg über Entwicklertools (Formularsteuerelemente) oder VBA bietet Excel seit geraumer Zeit **native** Checkboxes, die:

- direkt in eine Zelle eingefügt werden können,
- einen logischen Wahrheitswert TRUE/FALSE liefern,
- formelkompatibel sind; zum Beispiel mit WENN(), FILTER(), SORTIEREN(),
- in dynamischen Tabellen, Listen und Dashboards verwendet werden können,
- auf Mobilgeräten und im Web vollständig unterstützt werden.

## So fügen Sie ein Kontrollkästchen ein

So einfach geht's:

- Klicken Sie in die Zelle, in die Sie das Kontrollkästchen einfügen möchten.
- Wählen Sie im Menü **Einfügen** > **Steuerelemente** > **Kontrollkästchen** (englisch: „Checkbox“).
- Das Kontrollkästchen erscheint direkt in der Zelle. Es ist automatisch mit dem Zellwert verknüpft.



Kontrollkästchen (Checkbox) direkt in eine Excel-Zelle einfügen

Das Kontrollkästchen in der obigen Abbildung ist deaktiviert. Dementsprechend wird der Wert **FALSCH** in der Bearbeitungsleiste angezeigt. Er entspricht dem numerischen Wert , mit dem Sie im Weiteren rechnen können.

## Technischer Hintergrund

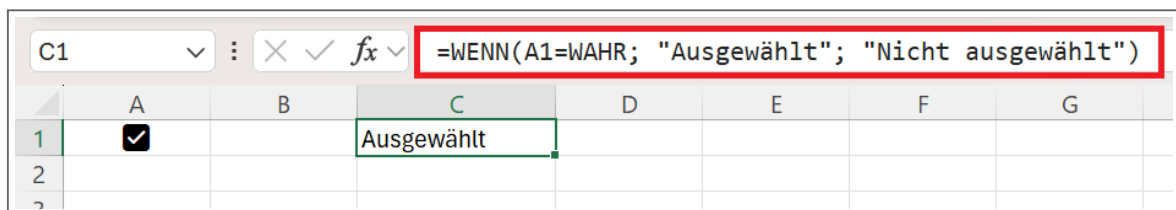
Ein gesetztes Kontrollkästchen gibt den Wert **WAHR** (TRUE) zurück, ein deaktiviertes **FALSCH** (FALSE). Dadurch lassen sich die Kontrollkästchen direkt in Berechnungen, Filterungen oder logischen Abfragen verwenden.

## Beispiele für praktische Anwendungen

### Einsatz in Funktionen oder Formeln

Sie haben in der Zelle A1 ein Kontrollkästchen eingefügt und werten dies nun in der Zelle C1 aus. Dazu geben Sie in C1 die Funktion ein:

**=WENN(A1=WAHR; "Ausgewählt"; "Nicht ausgewählt")**



*Auswertung des Kontrollkästchens mit WENN()*

### To-do-Liste mit automatischer Formatierung

Setzen Sie das Kontrollkästchen in To-do-Listen ein. So können Sie anschließend zum Beispiel auswerten, wie viele Aufgaben (To-dos) bereits erledigt sind:

**=ZÄHLENWENN(B2:B11; "WAHR")**

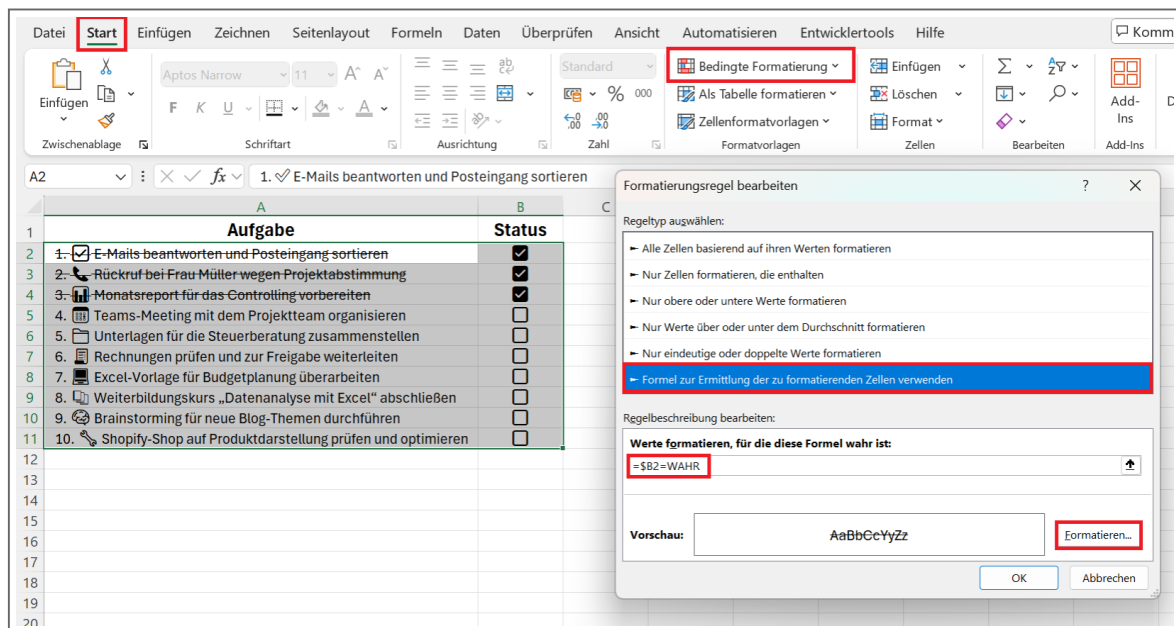


B13		=ZÄHLENWENN(B2:B11;"WAHR")
	A	B
1	Aufgabe	Status
2	1. <input checked="" type="checkbox"/> E-Mails beantworten und Posteingang sortieren	<input checked="" type="checkbox"/>
3	2. <input type="checkbox"/> Rückruf bei Frau Müller wegen Projektabstimmung	<input checked="" type="checkbox"/>
4	3. <input type="checkbox"/> Monatsreport für das Controlling vorbereiten	<input type="checkbox"/>
5	4. <input type="checkbox"/> Teams-Meeting mit dem Projektteam organisieren	<input checked="" type="checkbox"/>
6	5. <input type="checkbox"/> Unterlagen für die Steuerberatung zusammenstellen	<input type="checkbox"/>
7	6. <input type="checkbox"/> Rechnungen prüfen und zur Freigabe weiterleiten	<input type="checkbox"/>
8	7. <input type="checkbox"/> Excel-Vorlage für Budgetplanung überarbeiten	<input type="checkbox"/>
9	8. <input type="checkbox"/> Weiterbildungskurs „Datenanalyse mit Excel“ abschließen	<input type="checkbox"/>
10	9. <input type="checkbox"/> Brainstorming für neue Blog-Themen durchführen	<input type="checkbox"/>
11	10. <input type="checkbox"/> Shopify-Shop auf Produktdarstellung prüfen und optimieren	<input type="checkbox"/>
12		
13	Erledigt	3

Kontrollkästchen in einer Excel-To-Do-Liste

Verwenden Sie die Bedingte Formatierung, um erledigte Aufgaben durchzustreichen:

- Aktivieren Sie hierzu zunächst den Bereich der To-do-Liste in der vorigen Abbildung (A2:B11)
- Aktivieren Sie im Menüband die Befehlsfolge Registerkarte **Start** > Befehlsgruppe **Formatvorlagen** > Befehl **Bedingte Formatierung** > **Neue Regel**
- Wählen Sie im Dialogfeld **Formatierungsregel** den Eintrag **Formel zur Ermittlung der zu formatierenden Zellen verwenden**
- Erfassen Sie unter **Werte formatieren, für die diese Formel wahr ist:**  
**=\$B2=WAHR**
- Klicken Sie anschließend auf die Schaltfläche **Formatieren**.
- Es öffnet sich das Dialogfeld **Zellen formatieren**. Klicken Sie oben auf die Registerkarte **Schrift** und wählen Sie hier das Kontrollkästchen **Durchgestrichen** aus.
- Schließen Sie anschließend alle offenen Dialogfelder durch einen Klick auf **OK**.



Kontrollkästchen zur Steuerung der Bedingten Formatierung nutzen

Danach werden die erledigten Aufgaben durchgestrichen in der To-do-Liste dargestellt.

	A	B
1	Aufgabe	Status
2	1. <input checked="" type="checkbox"/> E-Mails beantworten und Posteingang sortieren	<input checked="" type="checkbox"/>
3	2. <input checked="" type="checkbox"/> Rückruf bei Frau Müller wegen Projektabstimmung	<input checked="" type="checkbox"/>
4	3. <input checked="" type="checkbox"/> Monatsreport für das Controlling vorbereiten	<input checked="" type="checkbox"/>
5	4. <input type="checkbox"/> Teams-Meeting mit dem Projektteam organisieren	<input type="checkbox"/>
6	5. <input type="checkbox"/> Unterlagen für die Steuerberatung zusammenstellen	<input type="checkbox"/>
7	6. <input type="checkbox"/> Rechnungen prüfen und zur Freigabe weiterleiten	<input type="checkbox"/>
8	7. <input type="checkbox"/> Excel-Vorlage für Budgetplanung überarbeiten	<input type="checkbox"/>
9	8. <input type="checkbox"/> Weiterbildungskurs „Datenanalyse mit Excel“ abschließen	<input type="checkbox"/>
10	9. <input type="checkbox"/> Brainstorming für neue Blog-Themen durchführen	<input type="checkbox"/>
11	10. <input type="checkbox"/> Shopify-Shop auf Produktdarstellung prüfen und optimieren	<input type="checkbox"/>

Ergebnis: Bedingte Formatierung mit Kontrollkästchen

## Dynamische Filterliste

Sie möchten noch nicht erledigte Aufgaben automatisch filtern und an eine andere Stelle ausgeben?

Verwenden Sie die FILTER()-Funktion, um nur die nicht erledigten Aufgaben zu extrahieren:

**=FILTER(A2:B11; B2:B11=FALSCH)**

	A	B	C	D	E
1	<b>Aufgabe</b>	<b>Status</b>			
2	1. E-Mails beantworten und Posteingang sortieren	<input checked="" type="checkbox"/>		4. Teams-Meeting mit dem Projektteam organisieren	FALSCH
3	2. Rückruf bei Frau Müller wegen Projektabstimmung	<input checked="" type="checkbox"/>		5. Unterlagen für die Steuerberatung zusammenstellen	FALSCH
4	3. Monatsreport für das Controlling vorbereiten	<input checked="" type="checkbox"/>		6. Rechnungen prüfen und zur Freigabe weiterleiten	FALSCH
5	4. Teams-Meeting mit dem Projektteam organisieren	<input type="checkbox"/>		7. Excel-Vorlage für Budgetplanung überarbeiten	FALSCH
6	5. Unterlagen für die Steuerberatung zusammenstellen	<input type="checkbox"/>		8. Weiterbildungskurs „Datenanalyse mit Excel“ abschließen	FALSCH
7	6. Rechnungen prüfen und zur Freigabe weiterleiten	<input type="checkbox"/>		9. Brainstorming für neue Blog-Themen durchführen	FALSCH
8	7. Excel-Vorlage für Budgetplanung überarbeiten	<input type="checkbox"/>		10. Shopify-Shop auf Produktdarstellung prüfen und optimieren	FALSCH
9	8. Weiterbildungskurs „Datenanalyse mit Excel“ abschließen	<input type="checkbox"/>			
10	9. Brainstorming für neue Blog-Themen durchführen	<input type="checkbox"/>			
11	10. Shopify-Shop auf Produktdarstellung prüfen und optimieren	<input type="checkbox"/>			
12					

*Filter-Funktion in Excel mit Kontrollkästchen steuern*

## Tipps zur Gestaltung

- Verwenden Sie Zellhintergrundfarben, um gesetzte Checkboxes visuell hervorzuheben (**Bedingte Formatierung**)
- Setzen Sie Kontrollkästchen in **intelligenten Tabellen** (STRG + T) ein, um dynamisch damit zu arbeiten.
- Nutzen Sie **VERWEIS-** oder **FILTER-Funktionen**, um Checklisten oder Feedback-Formulare auszuwerten.
- In Kombination mit **LET()** und **LAMBDA()** lassen sich sogar komplexe Steuerungen umsetzen.

## So haben Sie immer alle Excel-Makros sofort zur Verfügung

Wenn Sie mit Excel einige Makros immer wieder und für alle Excel-Dateien benötigen, dann sollten Sie diese in der persönlichen Makroarbeitsmappe abspeichern. Diese wird beim Öffnen von Excel immer als Pool der verfügbaren Makros geladen. Erfahren Sie, wie ein Makro in der Arbeitsmappe PERSONAL.XLSB abgespeichert wird.

Zuletzt geändert am 18.03.2026



Excel speichert alle Makros in sogenannten Modulen ab, die in den Arbeitsmappen (Dateien) integriert sind.

Damit Sie ein Makro im Menüband über die Schaltfläche **Makros** auswählen und starten können, müssen Sie allerdings die Arbeitsmappe, die den gewünschten Makro enthält, in Excel öffnen.

Dies ist bei Makros, die Sie häufig in der täglichen Praxis benötigen, umständlich. Viel besser wäre es, wenn alle regelmäßig benötigten **Makros automatisch immer zur Verfügung** stünden, wenn Excel geöffnet ist.

### Persönliche Makroarbeitsmappe anlegen

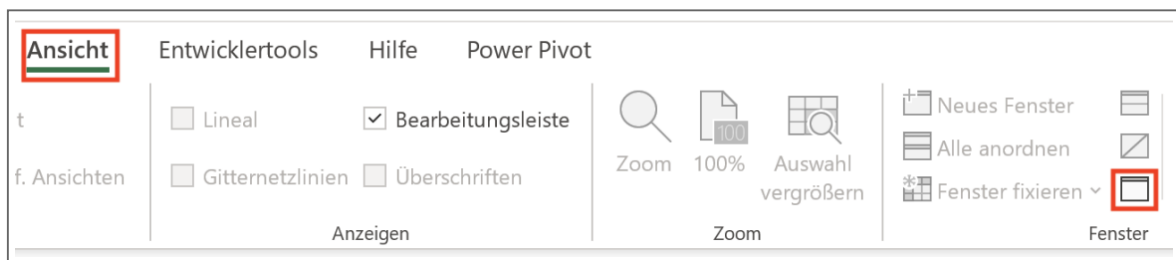
Für häufig benötigte Makros hat Microsoft die **persönliche Makroarbeitsmappe** in Excel integriert. Hierbei handelt es sich um eine Datei, die automatisch beim Starten von Excel geöffnet wird.

Alle in dieser Datei gespeicherten Programme stehen Ihnen dann sofort zum Einsatz zur Verfügung. Der Name dieser Datei lautet: **PERSONAL.XLSB**.

Excel blendet diese Arbeitsmappe standardmäßig aus. Das bedeutet, dass diese im Allgemeinen für Sie nicht sichtbar ist.

Vielleicht haben Sie oder ein Kollege in der Vergangenheit unbewusst eine persönliche Makroarbeitsmappe erstellt und wissen dies gar nicht. Mit einem einfachen Trick können Sie prüfen, ob auf dem Rechner eine persönliche Makroarbeitsmappe zum Einsatz kommt.

Aktivieren Sie hierzu im Menüband die Befehlsfolge Registerkarte **Ansicht** > Befehlsgruppe **Fenster** > Befehl **Fenster einblenden**.

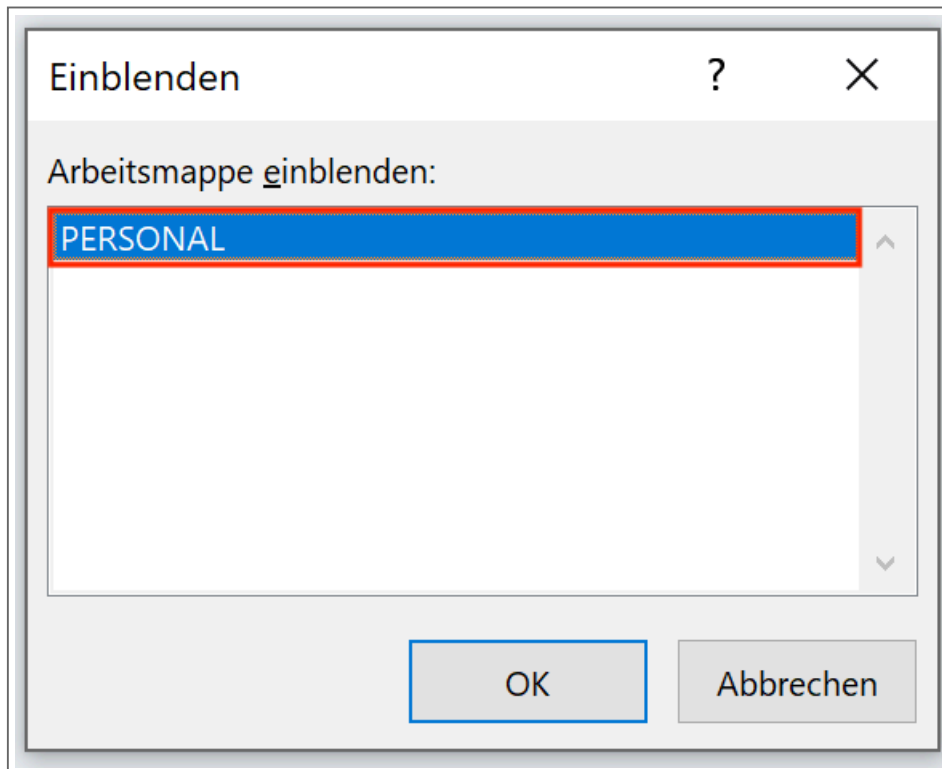


*Befehl Fenster einblenden aktivieren*

Daraufhin öffnet sich das Dialogfeld **Einblenden**. Hier bekommen Sie alle ausgeblendeten Arbeitsmappen aufgelistet.

Befindet sich in dieser Liste die Datei **PERSONAL.XLSB** oder **PERSONAL**, dann verwenden Sie bereits eine persönliche Makroarbeitsmappe, die automatisch beim Starten von Excel geöffnet wird.

Alle in dieser Datei gespeicherten Makros stehen Ihnen somit immer in Excel zur Verfügung. Sie müssen nicht mühsam erst die jeweilige Datei mit dem Makro suchen und öffnen.



Liste der ausgeblendeten Arbeitsmappen mit der persönlichen Makroarbeitsmappe

Wenn das Menü **Fenster Einblenden** ausgegraut ist, gibt es keine Tabellen oder Arbeitsmappen im Hintergrund. Es existiert dann auch noch keine Datei PERSONAL.XLSB mit Makros.

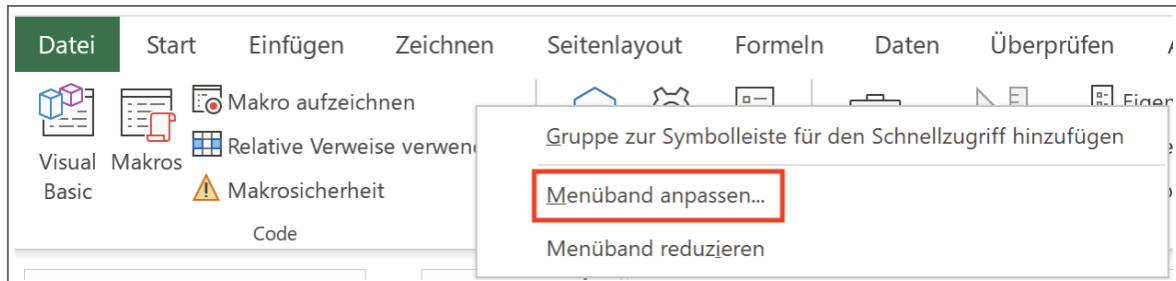
## Registerkarte Entwicklertools einblenden und nutzen

Sollten Sie noch keine persönliche Makroarbeitsmappe erstellt haben, dann können Sie diese ohne große Probleme selbst erstellen.

Davor sollten Sie prüfen, ob in Ihrem Menüband die Registerkarte **Entwicklertools** eingeblendet ist. In dieser Registerkarte finden Sie standardmäßig viele nützliche Befehle und Tools rund um das Thema Makro und VBA in Excel.

Leider ist diese nützliche Registerkarte in Excel oft ausgeblendet.

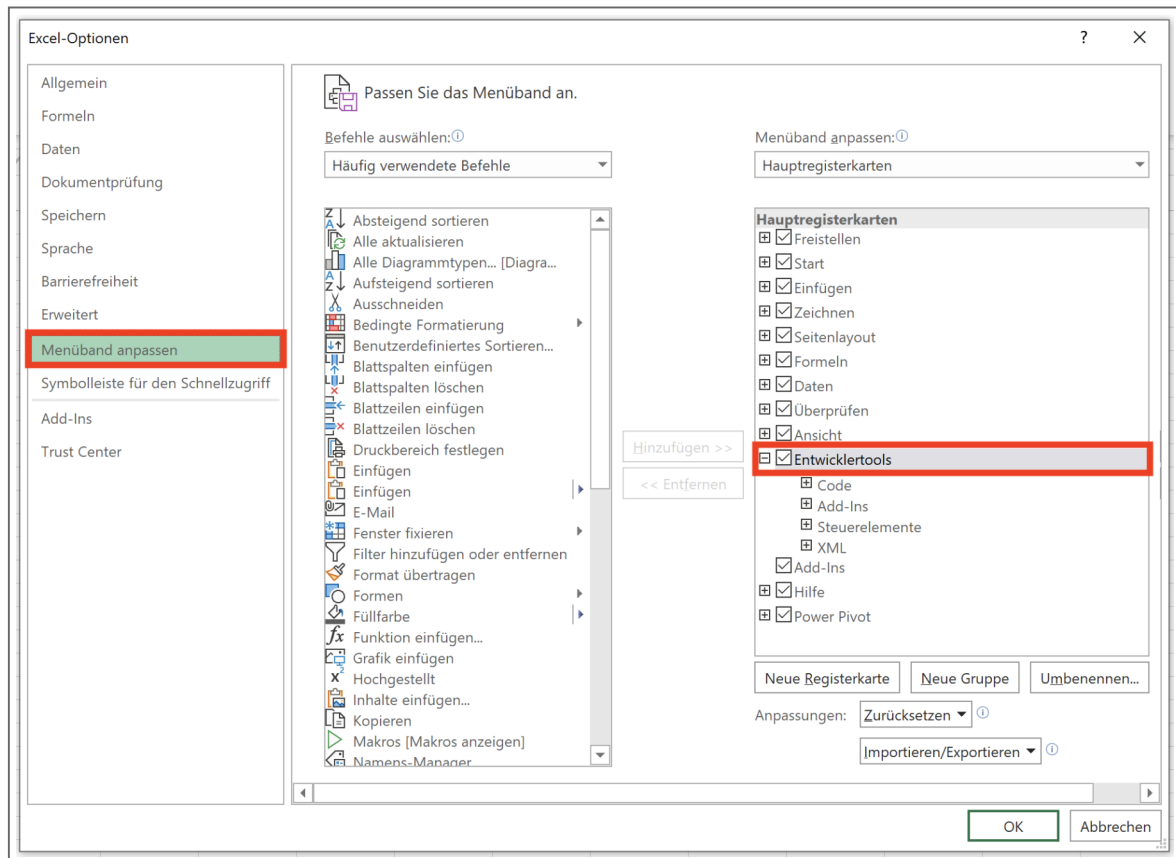
Sollten Sie die Registerkarte **Entwicklertools** im Menüband nicht finden, dann klicken Sie mit der rechten Maustaste auf eine beliebige Stelle im Menüband und wählen anschließend im Kontextmenü den Eintrag **Menüband anpassen...** aus.



*Befehl Menüband anpassen... im Kontextmenü auswählen*

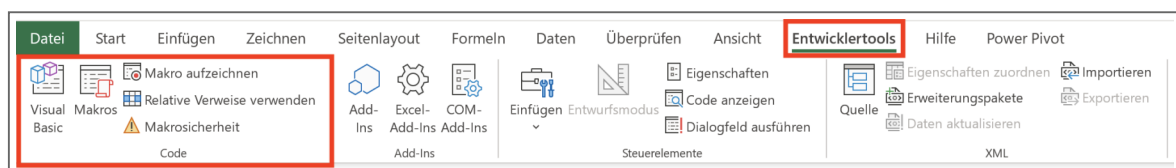
Es öffnet sich das Dialogfeld **Excel-Optionen**. Am linken Rand ist hier standardmäßig der Eintrag **Menüband anpassen** ausgewählt.

Aktivieren Sie in der rechten Liste das Kontrollkästchen für die Registerkarte **Entwicklertools** und bestätigen Sie Ihre Einstellung, indem Sie das Dialogfeld durch einen Klick auf **OK** schließen.



Registerkarte *Entwicklertools* im Menüband aktivieren

Die Registerkarte **Entwicklertools** wird daraufhin im Menüband dargestellt. Sie finden die wichtigsten Befehle für Makros hier am linken Rand unter der Befehlsgruppe **Code**.



Registerkarte *Entwicklertools* im Menüband

## So einfach können Sie eine persönliche Makroarbeitsmappe erstellen

Am einfachsten erstellen Sie eine persönliche Makroarbeitsmappe, indem Sie ein Makro in der persönlichen Makroarbeitsmappe aufzeichnen. Durch diese Aktion wird automatisch in Excel die persönliche Makroarbeitsmappe angelegt.



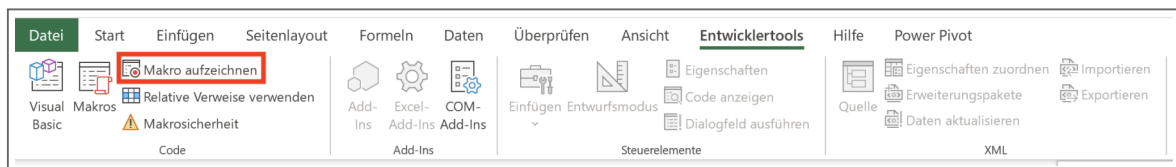
## Beispiel für ein einfaches Makro mit Excel

Wie das genau funktioniert, soll im Folgenden am Beispiel eines Makros gezeigt werden, das folgende Befehlsfolge automatisch durchführt, wenn es gestartet wird:

- Die aktive Arbeitsmappe wird gespeichert (**Strg + S**).
- Die aktive Arbeitsmappe wird geschlossen (**Strg + W**).

Öffnen Sie zunächst eine beliebige Arbeitsmappe, die Sie schon einmal abgespeichert haben.

Aktivieren Sie jetzt im Menüband die Befehlsfolge Registerkarte **Entwicklertools** > Befehlsgruppe **Code** > Befehl **Makro aufzeichnen**. Daraufhin öffnet sich das Dialogfeld **Makro aufzeichnen**.



*Befehl Makro aufzeichnen ausführen*

## Das Makro beschreiben

Vergeben Sie hier unter **Makroname** einen aussagekräftigen Namen.

**Wichtig:** Sie können hier keine Leerzeichen und Bindestriche einsetzen. Verwenden Sie am besten als Alternative hierfür Unterstriche. Das erste Zeichen muss immer ein Buchstabe sein. Als weitere Zeichen dürfen Sie auch Zahlen verwenden.

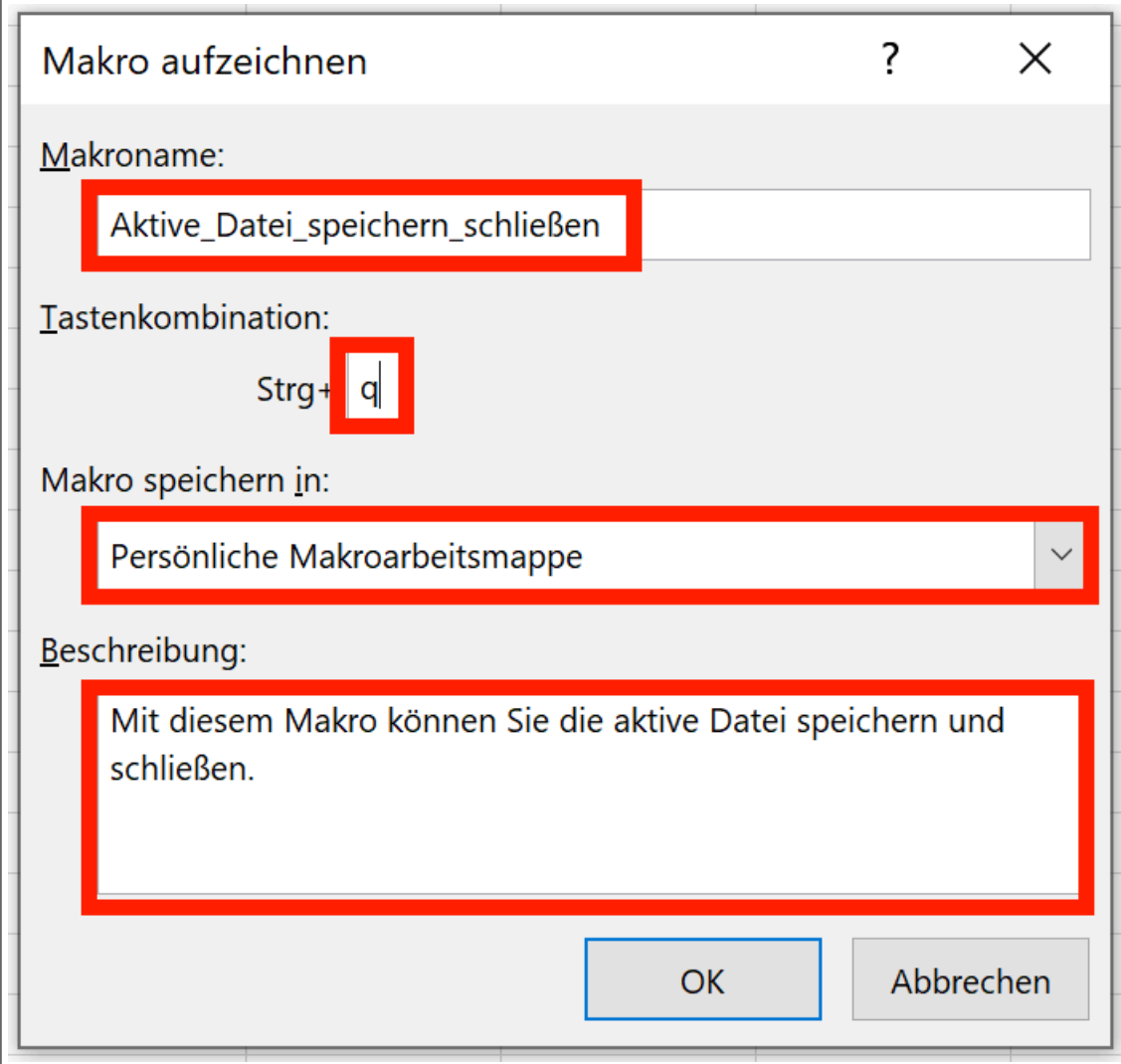
Der Beispielsname für dieses Makro sei **Aktive\_Datei\_speichern\_schließen**.

Unter **Tastenkombination** können Sie ein Zeichen ergänzen. Wenn Sie diese Tastenkombination anschließend nach dem Aufzeichnen drücken, dann wird das Makro automatisch mit dieser Tastenkombination gestartet.

Über das Listenfeld **Makro speichern in** definieren Sie, in welcher Arbeitsmappe das Makro gespeichert werden soll. Wählen Sie hier jetzt den Eintrag **Persönliche Makroarbeitsmappe** aus.

Durch diese Auswahl wird automatisch beim Aufzeichnen die persönliche Makroarbeitsmappe angelegt.

Unter **Beschreibung** können Sie noch einen Kommentar oder eine Beschreibung zu Dokumentationszwecken dem Makro hinzufügen.



Makro aufzeichnen

Makroname: Aktive\_Datei\_speichern\_schließen

Tastenkombination: Strg+ q

Makro speichern in: Persönliche Makroarbeitsmappe

Beschreibung: Mit diesem Makro können Sie die aktive Datei speichern und schließen.

OK Abbrechen

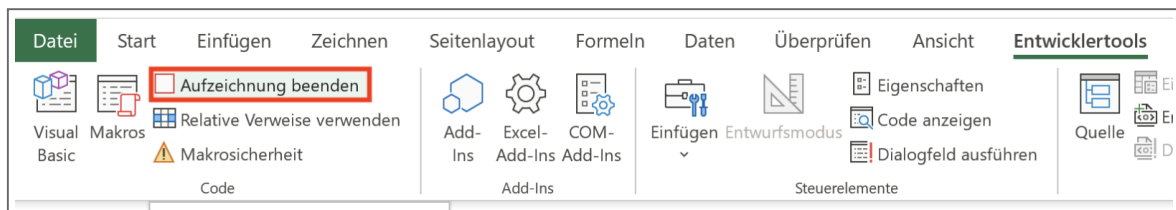
Dialogfeld Makro aufzeichnen

## Makro aufzeichnen

Starten Sie jetzt die Aufzeichnung, indem Sie das Dialogfeld durch einen Klick auf **OK** schließen. Ab jetzt werden alle Aktionen und Befehle, die Sie ausführen, automatisch von Excel aufgezeichnet und in ein VBA-Programm (Makro) umgewandelt.

Führen Sie daher jetzt die folgenden Aktionen durch:

- Drücken Sie die Tastenkombination **Strg + S**. Durch diese Aktion wird die aktive Arbeitsmappe gespeichert.
- Drücken Sie die Tastenkombination **Strg + W**. Durch diese Aktion wird die aktive Arbeitsmappe geschlossen.
- Beenden Sie jetzt die Aufzeichnung, indem Sie im Menüband die Befehlsfolge Registerkarte **Entwicklertools** > Befehlsgruppe **Code** > Befehl **Aufzeichnung beenden** ausführen.



*Makro-Aufzeichnung beenden*

## Erstellte Makros ansehen

Das erstellte Makro in der persönlichen Makroarbeitsmappe können Sie sich jetzt anzeigen lassen. Klicken Sie im Menüband auf die Befehlsfolge Registerkarte **Entwicklertools** > Befehlsgruppe **Code** > Befehl **Visual Basic**.



*Befehl Visual Basic aktivieren*

Durch diese Aktion öffnen Sie den Visual-Basic-Editor (VBE). Sie können den Editor auch durch die Tastenkombination **Alt + F11** öffnen.

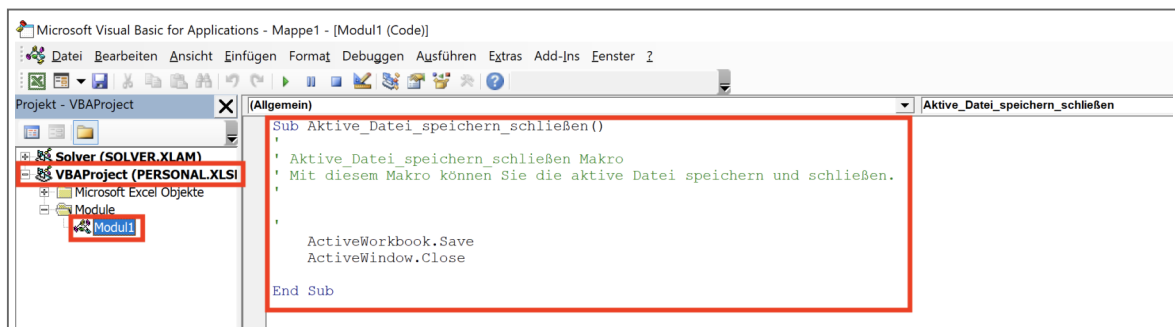
Hier bekommen Sie alle Makros dargestellt, die sich in geöffneten Dateien befinden. Im **Projektextplorer** am linken Rand bekommen Sie die einzelnen Projekte dargestellt. Ein Projekt entspricht einer Arbeitsmappe.

Hierarchisch darunter bekommen Sie die Ordner für **Microsoft Excel Objekte** und **Module** dargestellt.

Öffnen Sie unter dem Projekt **PERSONAL.XLSB** den Ordner für **Module**. Hier finden Sie – analog zu den Tabellenblättern in den Arbeitsmappen – Module, in denen die Makros gespeichert werden.

Im Beispiel ist nur ein Modul vorhanden, da erst ein Makro aufgezeichnet wurde.

Klicken Sie hier auf den Eintrag **Modul1**. Anschließend erhalten Sie im Editor rechts daneben alle in diesem Modul gespeicherten Makros dargestellt.



*Makro Datei\_speichern\_schließen im Visual Basic Editor*

Schließen Sie nun den Visual-Basic-Editor durch einen Klick auf das **X** am rechten oberen Rand des Fensters. Alternativ können Sie den Visual-Basic-Editor auch schließen, indem Sie die Tastenkombination **Alt + Q** drücken.

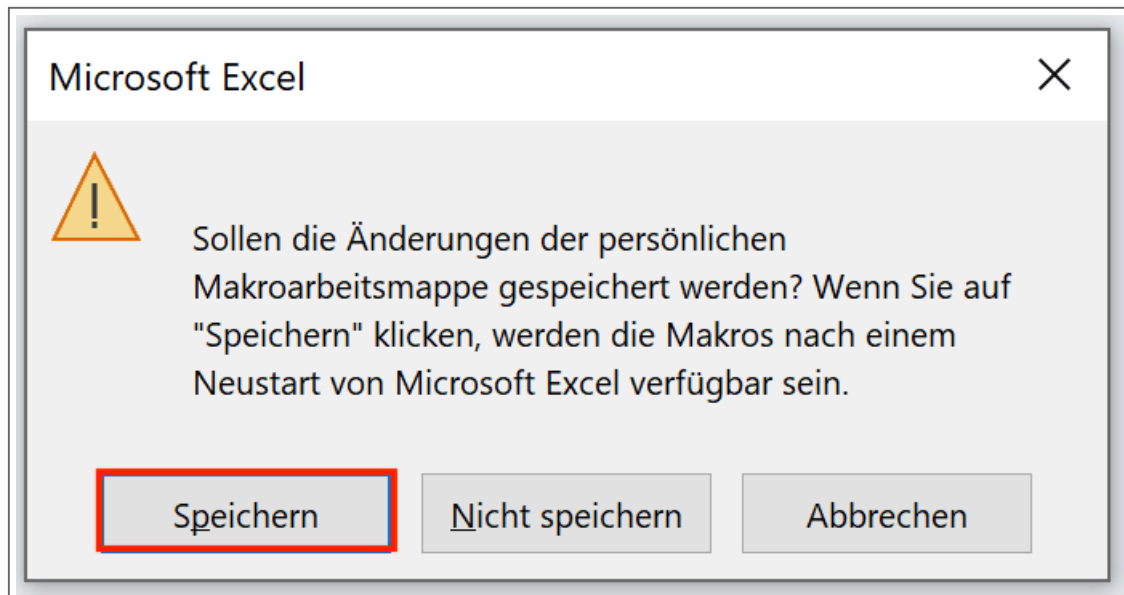
## Zum Schluss noch die persönliche Makroarbeitsmappe speichern

Sie gelangen nun wieder zu der gewohnten Excel-Oberfläche zurück. Sie haben jetzt ein Makro in der persönlichen Makroarbeitsmappe erstellt.

Die persönliche Makroarbeitsmappe ist eine gewöhnliche Excel-Datei, die im Moment aber noch nicht gespeichert ist. Damit die Makros auch später zur Verfügung stehen, müssen Sie diese **Datei speichern**, sonst kann das Makro beim erneuten Starten von Excel nicht gefunden werden.

Beenden Sie jetzt einfach Excel, indem Sie in der rechten oberen Ecke des Excel-Fensters, auf das **X** klicken. Wenn Sie Excel beenden und wenn noch nicht alle Dateien gespeichert sind, dann fragt Excel standardmäßig nach, ob Sie diese Dateien abspeichern wollen.

Ihnen wird daher ein Dialogfeld angezeigt, in dem Sie gefragt werden, ob Sie die Änderung der persönlichen Makroarbeitsmappe speichern wollen. Klicken Sie hier auf die Schaltfläche **Speichern**.

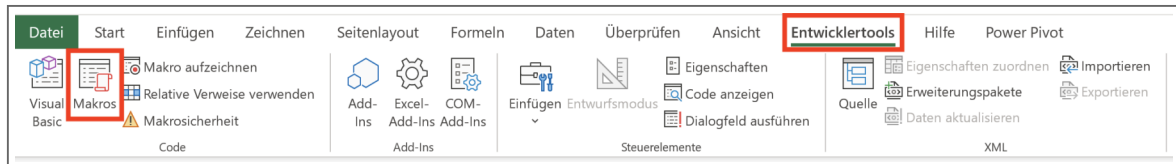


*Persönliche Makroarbeitsmappe speichern*

## Alle wichtigen Makros beim Start von Excel verfügbar haben

Wenn Sie Excel jetzt neu starten, dann wird die persönliche Makroarbeitsmappe automatisch geöffnet und Sie können sofort auf das erstellte Makro zugreifen.

Aktivieren Sie im Menüband die Befehlsfolge Registerkarte **Entwicklertools** > Befehlsfolge **Code** > Befehl **Makros**. Alternativ können Sie die Tastenkombination **Alt + F8** verwenden.

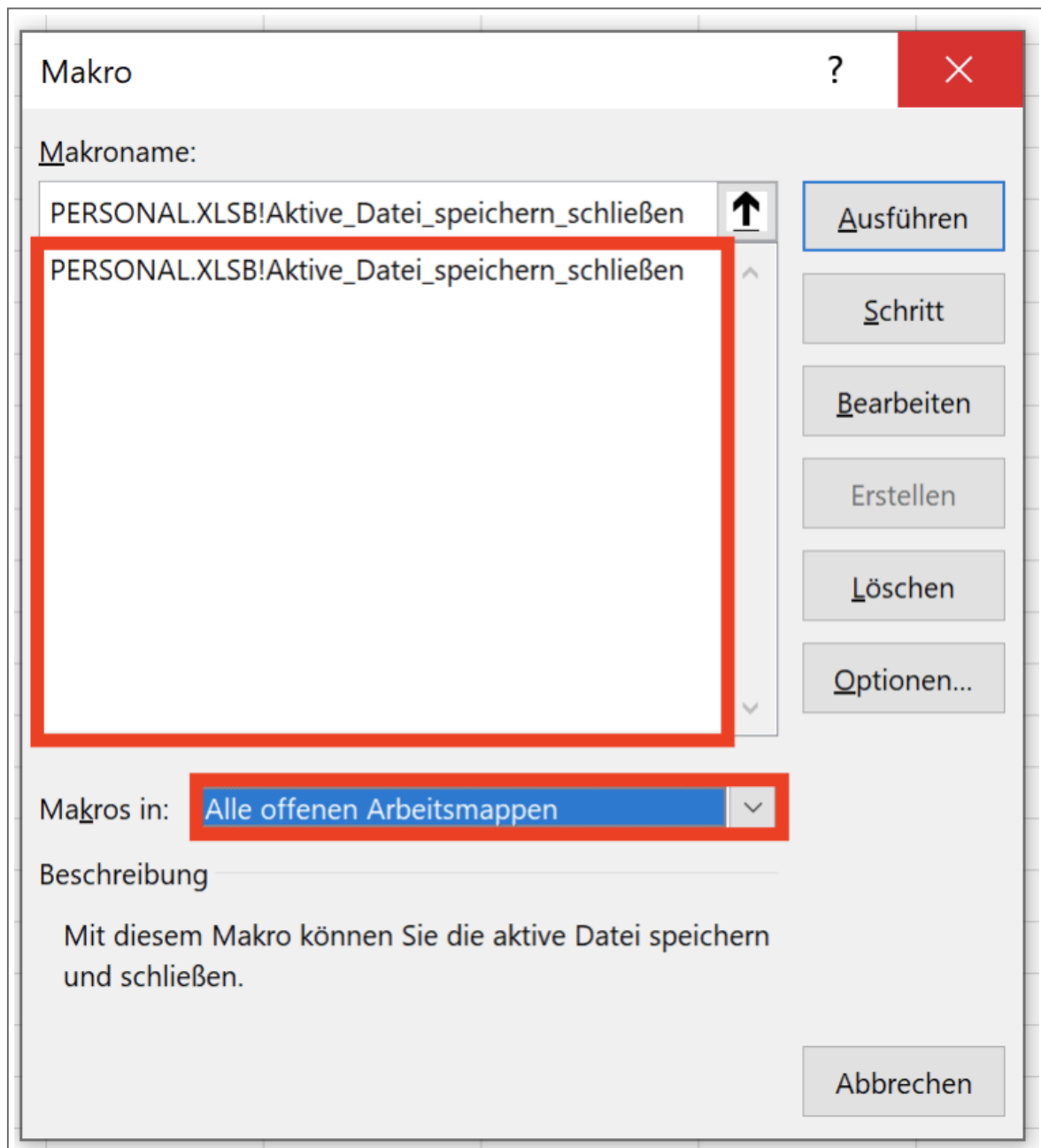


*Befehl Markos starten*

Es öffnet sich das Dialogfeld **Makro**. Hier bekommen Sie alle verfügbaren Makros aufgelistet.

Per Doppelklick auf den jeweiligen Namen können Sie das entsprechende Makro starten. Sie können alternativ die Tastenkombination **Strg + Q** verwenden, die Sie beim Aufzeichnen des Makros definiert haben.

Obwohl Sie keine andere Datei geöffnet haben, steht der eben erstellte Makro sofort zur Verfügung.



Dialogfeld Makro

## Sammlung häufig benötigter Makros erstellen

Kopieren Sie jetzt alle Makros, die Sie häufig benötigen, in den Visual-Basic-Editor und hier in das Modul der persönlichen Makroarbeitsmappe, damit Sie immer und sofort auf diese zugreifen können.

Öffnen Sie dazu die Excel-Datei, in der sich das Makro befindet, das Sie in Ihre PERSONAL.XLSB kopieren möchten.

Klicken Sie im Menüband auf die Befehlsfolge Registerkarte **Entwicklertools** > Befehlsgruppe **Code** > Befehl **Visual Basic**.

Im Visual-Basic-Editor sehen Sie in der linken Spalte die Datei PERSONAL.XLSB und die Datei mit Ihrem Makro. Ziehen Sie aus Ihrer Datei das Modul mit Ihrem Makro in die Datei **VBAProject (PERSONAL.XLSB) > Module**.

Schließen Sie alle offenen Excel-Dateien und bestätigen Sie dabei, dass die Datei PERSONAL.XLSB gespeichert wird.



## Blattschutz mit Excel-Makro aktivieren

Mit VBA kann der Blattschutz in einer Excel-Arbeitsmappe gesteuert werden. Mit diesen Makro-Befehlen lässt sich für alle Tabellen auf einmal der Blattschutz einschalten und ausschalten.

Zuletzt geändert am 18.03.2026



Mit VBA lässt sich der Blattschutz von Tabellen setzen und aufheben:

- Der VBA-Befehl in Excel zum Setzen des Blattschutzes heißt **Worksheet.Protect**.
- Der Befehl zum Aufheben des Schutzes heißt **Worksheet.Unprotect**.

Die Protect-Methode wirkt sich nur auf ungeschützte Arbeitsblätter aus. Bei bereits geschützten Arbeitsblättern bleibt die Protect-Methode wirkungslos.

## Makro für Blattschutz ohne Kennwort

Mit den folgenden Makros wird der Blattschutz für Tabelle1 ohne Kennwort gesetzt und aufgehoben.

```
Sub Blattschutz_ein()  
  Sheets("Tabelle1").Protect  
End Sub  
  
Sub Blattschutz_aus()  
  Sheets("Tabelle1").Unprotect  
End Sub
```

## Makro für Blattschutz mit Kennwort

Der Blattschutz kann auch mit Kennwort gesetzt werden. Das Kennwort lautet im folgenden Beispiel „meinPasswort“.

```
Sub Blattschutz_ein_mit_Passwort()  
  Sheets("Tabelle1").Protect Password:="meinPasswort"  
End Sub  
  
Sub Blattschutz_aus_mit_Passwort()  
  Sheets("Tabelle1").Unprotect Password:="meinPasswort"  
End Sub
```

## Blattschutz für alle Tabellen der Arbeitsmappe

Mit einer For-Schleife wird der Blattschutz für alle Tabellen einer Arbeitsmappe eingeschaltet oder aufgehoben. Im folgenden Beispiel wird der Blattschutz ohne Kennwort verwendet.

```
Sub Blattschutz_ein_alle_Tabellen()  
  For Each sheet In ActiveWorkbook.Worksheets  
    sheet.Protect  
  Next sheet  
End Sub  
  
Sub Blattschutz_aus_alle_Tabellen()  
  For Each sheet In ActiveWorkbook.Worksheets  
    sheet.Unprotect  
  Next sheet  
End Sub
```

Nutzen Sie als  
**Premium-Mitglied**  
alle  
**Handbuch-Kapitel**  
mit mehr als  
3.000 Checklisten und Excel-Vorlagen

**Jetzt anmelden**

[www.business-wissen.de/anmelden/](http://www.business-wissen.de/anmelden/)

## Impressum

b-WISE GmbH Business Wissen Information Service  
Bismarckstraße 21  
76133 Karlsruhe  
DEUTSCHLAND

[service@business-wissen.de](mailto:service@business-wissen.de)  
Telefon +49 721 18397-0

Copyright 2026, b-wise GmbH, All Rights Reserved