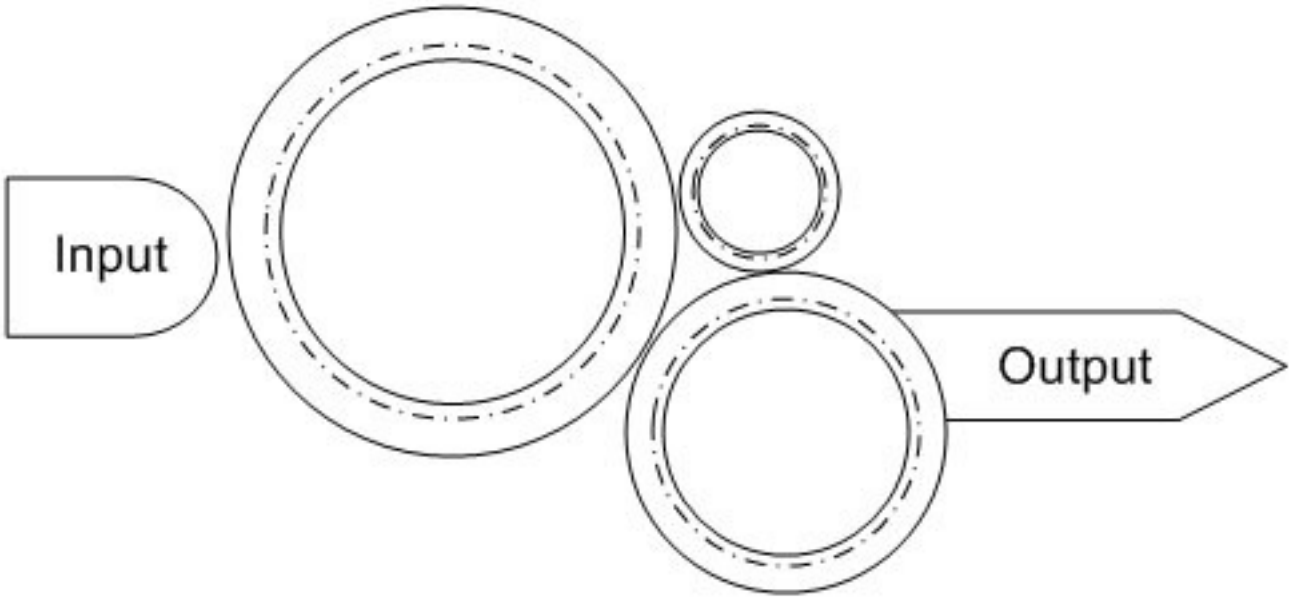
Excel – VBA
Prozeduren und Funktionen



Unterprogramme (Subroutinen)

- ... lösen Teilprobleme der Gesamtaufgabe.
- ... fassen Anweisungen, die ein bestimmtes Thema bearbeiten, zu einem Block zusammen.
- ... sind eine Abfolge von VBA-Befehlen und -Anweisungen, die zeilenweise abgearbeitet werden.
- ... fassen Code zusammen. Der Code wird an verschiedenen Stellen aufgerufen.
- ... werden Prozeduren genannt, falls sie keinen Wert an den Aufrufer zurückgeben.
- ... werden Funktionen genannt, falls sie einen Wert an den Aufrufer zurückgeben.

Merkmale

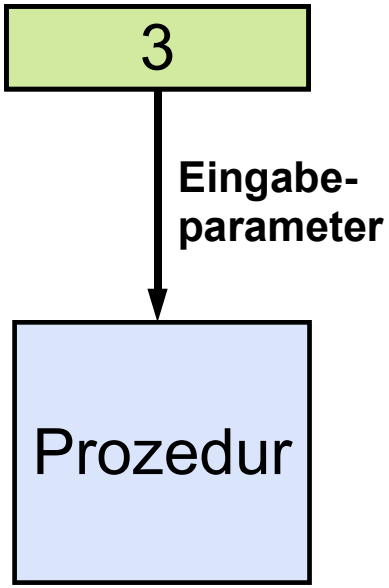
- Jedes Unterprogramm hat eine eindeutige Bezeichnung mit deren Hilfe es aufgerufen wird.
- Jedes Unterprogramm ist unabhängig von anderen Unterprogrammen. Ausnahme: Es ruft ein weiteres Unterprogramm auf und nutzt den Rückgabewert.
- Innerhalb eines Unterprogramms
  - ... kann ein weiteres Unterprogramm aufgerufen werden.
  - ... kann aber kein weiteres Unterprogramm deklariert werden.

Vorteile

- Die Aufgabenstellung wird in kleinere unabhängige Module eingeteilt und somit auch strukturiert
- Der Quellcode lässt sich besser lesen.
- Der Code einer Funktion kann in anderen Programmen wiederverwendet werden.
- Wiederholende Aufgaben werden in einer eigenständigen Prozedur gekapselt. Die gekapselte Prozedur kann von verschiedenen Stellen aufgerufen werden.
- Der Code muss nur an einer Stelle bearbeitet werden. Fehler lassen sich schneller finden.
- Veränderungen lassen sich einfacher vornehmen und testen. Es sind nur Codefragmente betroffen.

Wie arbeitet eine Prozedur?

- In einer Prozedur wird eine bestimmte Aufgabe gelöst. Der Nutzer der Prozedur muss nicht wissen, wie die Aufgabe gelöst wird. Dem Nutzer genügt es zu wissen, wie die Prozedur aufgerufen wird. Die Prozedur ist eine Blackbox.
- Der Prozedur können Parameter übergeben werden. Dem Nutzer der Prozedur muss der Typ der Parameter bekannt sein. Einige Prozeduren besitzen keine Eingabeparameter.



Beispiel: Zeilen und Spalten einer Auswahl

```
Sub Main()
 ...
    bereich = Selection.Address
    bereich = Replace(bereich, "$", "")
    spalte = Left(bereich, 1)

    pos = InStr(bereich, ":")
    txtZeile = Mid(bereich, anfangZeile, (pos - anfangZeile))
    zeile = CInt(txtZeile)

    txtEndZeile = Mid(bereich, (pos + anfangZeile))
    endZeile = CInt(txtEndZeile)
     Call SummeZelle(spalte, zeile, (endZeile - zeile))
End Sub
```

Prozeduren

- ... beginnen mit Sub und enden mit End Sub.
- ... fassen Anweisungen, die zu einer Aktion gehören zusammen.
- ... haben einen eindeutigen, frei wählbaren Namen.
- ... geben keinen Wert an den Aufrufer zurück.
- ... können aber Werte übergeben bekommen. Innerhalb der runden Klammern stehen alle Parameter, die einer Prozedur übergeben werden. Zwischen dem Prozedurnamen und der Klammer befindet sich kein Leerzeichen!

Prozeduren aufrufen

Call SummeZelle(spalte, zeile, (endZeile – zeile))

SummeZelle spalte, zeile, (endZeile – zeile)

- Die Prozedur SummeZelle wird aufgerufen.
- Der Prozedur werden Parameter durch den Aufrufer übergeben.
- Falls die Prozedur
  - ... mit Hilfe des Schlüsselworts Call aufgerufen wird, müssen die Parameter geklammert werden.
  - ... nur mit ihren Namen aufgerufen werden, dürfen die Parameter nicht geklammert werden.

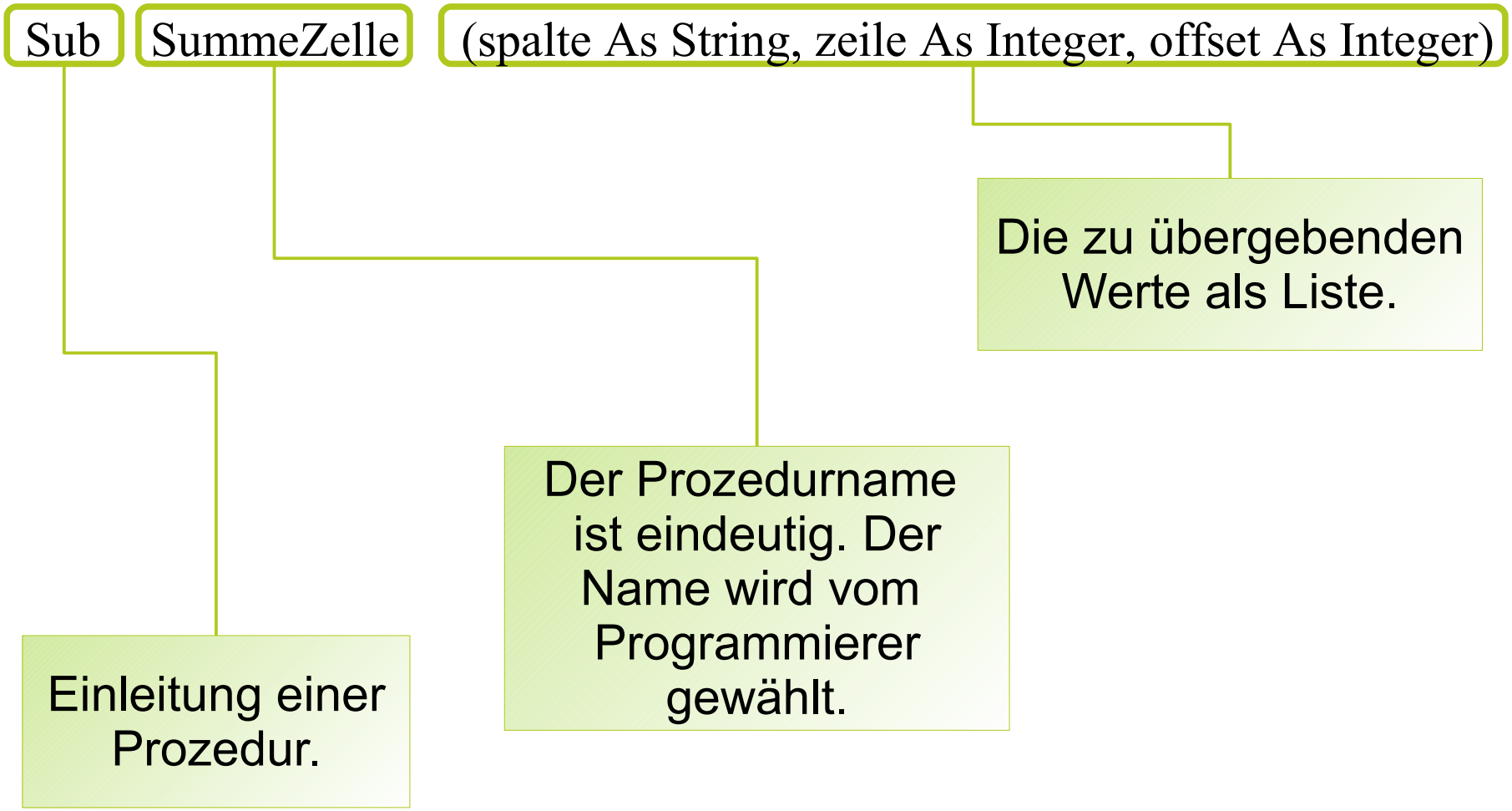
Prozeduren in VBA erstellen

```
Sub SummeZelle(spalte As String, zeile As Integer, offset As Integer)

    Range(spalte & (zeile + offset + 1)).Activate

    ActiveCell.Formula = "= SUM(" & spalte & zeile & ":"
                                  & spalte & (zeile + offset) & ")"
End Sub
```

Prozeduren bestehen aus

- ... dem Prozedurkopf `Sub Prozedur(parameter01, parameter02)`.
  - Schnittstelle nach außen.
  - Wie wird die Funktion aufgerufen?
- ... dem Prozedurrumpf.
  - Auszuführende Anweisungen.
- Jede Prozedur endet mit `End Sub`.

Prozedurkopf



Der Prozedurname

- ... beginnt immer mit einem Buchstaben.
- ... ist kürzer als 256 Zeichen.
- ... darf im Namen kein Leerzeichen, Punkt, Ausrufezeichen, Bindestrich, oder @, &, $ oder # haben.
- ... sollte keine Umlaute, Satzzeichen oder Sonderzeichen enthalten.
- Es gibt keine Unterscheidung zwischen der Groß- und Kleinschreibung von Namen.
- Schlüsselworter oder vordefinierte Prozedurnamen dürfen nicht genutzt werden.
- ... darf nur einmal vorkommen.
- ... sollte die Aufgabe der Prozedur widerspiegeln.

Hinweise

- Wenn die Aufgabe der Prozedur beschrieben werden soll, werden häufig Substantive genutzt. Welches Problem beschreibt die Prozedur? Beispiele:
  - Mittelwert(), Integral(), TuermeVonHanoi(), FehlerAusgeben().
- Adjektive werden für die Beschreibung von logischen Operationen genutzt. Beispiele:
  - kleinsterWert(), groessterMesswert().

Hinweise

- Wenn Berechnungen oder die Frage "Wie wird die Aufgabe erledigt?" beschrieben wird, werden häufig Verben genutzt. Beispiele:
  - intAddieren beschreibt eine Funktion die Ganzzahlen addiert.
  - Das Verb set... wird für Prozeduren genutzt, die Eigenschaften eines Objekts verändern.
  - Das Verb get... wird für Prozeduren genutzt, die den Eigenschaftswert eines Objekts lesen.
- Die Bezeichnung sollten nur einen Sprachraum nutzen. Beispiele:
  - main(), printOut(), add().
  - start(), drucken(), addieren().

Parameterliste

Sub [prozedur]([para01], [para02], ...)

- Die Liste wird mit Hilfe der runden Klammern begrenzt.
- Die Liste kann beliebig viele Parameter aufnehmen.
- Die einzelnen Listenelemente werden durch Kommata getrennt.
- Die Liste kann leer sein.
- Zwischen dem Prozedurnamen und dem Beginn der Parameterliste steht kein Leerzeichen.

Parameter deklarieren

```
Sub SummeZelle (spalte As String, zeile As Integer, offset As Integer)
```

Der Name der Variablen ist frei wählbar. Die Variable kann innerhalb der Prozedur genutzt werden.

Der Datentyp legt die Art des zu speichernden Wertes und deren Speicherbedarf fest.

Prozeduren aufrufen

```
Sub Main()
 ...

   txtEndZeile = Mid(bereich, (pos + anfangZeile))
   endZeile = CInt(txtEndZeile)
   SummeZelle spalte, zeile, (endZeile - zeile)
End Sub
```

Eine Prozedur wird durch den Namen aufgerufen.

Möglichkeiten für den Prozeduraufruf

SummeZelle spalte, zeile, (endZeile – zeile)

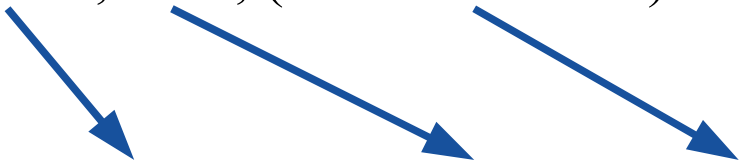
- Die Prozedur wird mit dem Namen aufgerufen.
- Die zu übergebenden Parameter werden durch Kommata getrennt.
- Der Prozedurname wird von der Parameterliste durch ein Leerzeichen getrennt.

Call SummeZelle(spalte, zeile, (endZeile – zeile))

- Der Aufruf wird mit Hilfe des Schlüsselwortes Call eingeleitet.
- Dem Schlüsselwort folgt der Prozedurname.
- Die Parameterliste muss geklammert werden.

Parameter zu ordnen

Aufruf: SummeZelle spalte, zeile, (endZeile - zeile)



Prozedur:

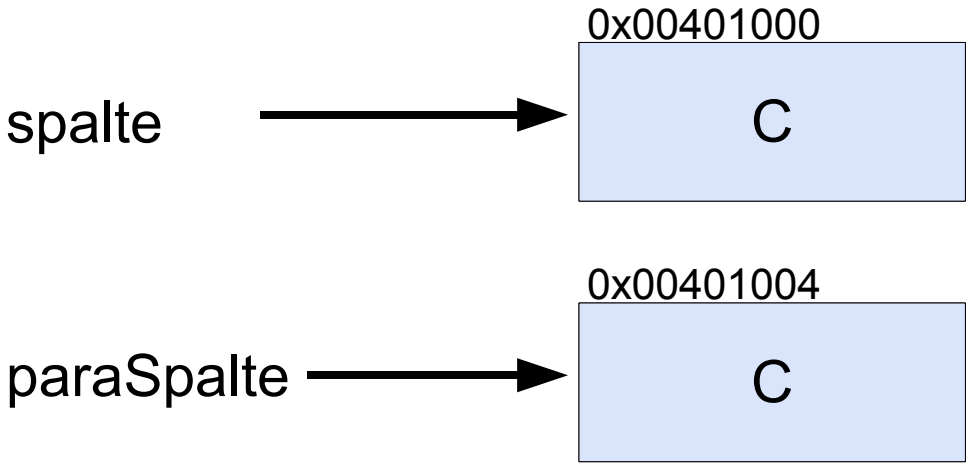
```
Sub SummeZelle(spalte As String, zeile As Integer, offset As Integer)
```

- Der erste Parameter im Aufruf wird dem ersten Parameter im Prozedurkopf zugeordnet und so weiter.
- Der Parameter im Prozedurkopf und die Variable im Aufruf haben den gleichen Datentyp.
- Aufruf und Prozedurkopf haben normalerweise die gleiche Anzahl von Parametern.

Parameterübergabe "Call By Value"

ByVal variable As Integer

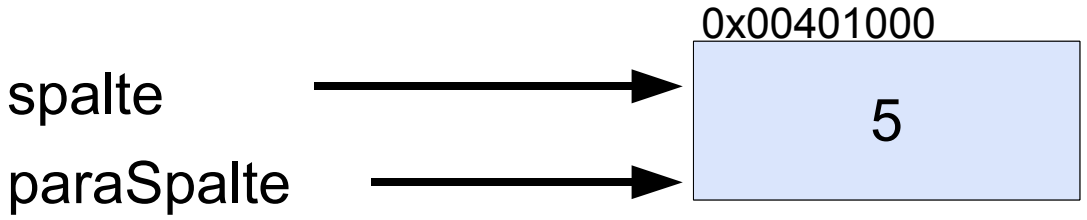
- Der Wert des Aufrufers wird in die Variable kopiert.
- Die Prozedur hat keinen Zugriff auf die Variable des Aufrufers.
- Die Variable in der Parameterliste und die Variable des Aufrufers haben den gleichen Wert, liegen aber in unterschiedlichen Schubladen (Speicherstellen).
- Schreibschutz für die zu übergebene Variable.



Parameterübergabe "Call By Reference"

ByRef variable As Integer

- Die Variable selber wird übergeben.
- Es wird ein Verweis auf einen Wert übergeben.
- Die Variable in der Parameterliste und die Variable des Aufrufers zeigen auf die gleiche Schublade (Speicherstellen).
- Der Wert der übergebenden Variablen kann durch die Prozedur verändert werden.
- Standardmäßig werden in VBA die Parameter als Verweis übergeben.



Optionale Parameter

```
Sub Addition(zahl01 As Integer, zahl02 As Integer, _
                    Optional zahl03 As Integer = 1)
    Dim summe As Integer

    summe = zahl01 + zahl02 + zahl03
    Debug.Print summe

End Sub

Sub Main()
    Call Addition(2, 3)
    Call Addition(2, 3, 4)
End Sub
```

Hinweise

- Optionale Parameter müssen beim Aufruf nicht übergeben werden.
- Mit Hilfe des Gleichheitszeichen kann optionalen Parametern ein Standardwert zugewiesen werden.
  - Datentypen für Ganz- oder Dezimalzahlen haben einen Standardwert von 0.
  - Strings sind standardmäßig mit einem leeren String initialisiert.
- Alle Nachfolger eines optionalen Parameters müssen auch optional sein.

Mit der Parameterliste arbeiten

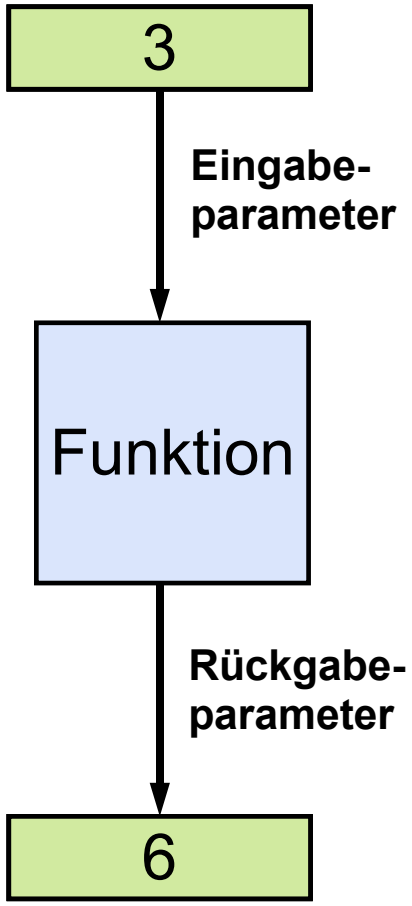
```
newText = Replace(baeren, oldZeichen, newZeichen, , , _
                                    vbTextCompare)
```

- Lange Parameterlisten können mit Hilfe des Unterstrich in der nächsten Zeile fortgesetzt werden. Vor dem Unterstrich muss ein Leerzeichen stehen.
- Jeder optionale Parameter kann weggelassen werden. Für jeden ausgelassenen Parameter muss ein Komma gesetzt werden. Das heißt, das dazugehörige Trennzeichnen muss gesetzt werden. Andere Möglichkeit:

```
newText = Replace(expression:=baeren, Find:=oldZeichen, _
                                    Replace:=newZeichen, Compare:=vbTextCompare)
```

Wie arbeitet eine Funktion?

- In einer Funktion wird eine bestimmte Aufgabe gelöst. Der Nutzer der Funktion muss nicht wissen, wie die Aufgabe gelöst wird. Dem Nutzer genügt es zu wissen, wie die Funktion aufgerufen wird. Die Funktion ist eine Blackbox.
- Der Funktion können Parameter übergeben werden. Dem Nutzer der Funktion muss die Art oder der Typ der Parameter bekannt sein. Einige Funktionen besitzen keine Eingabeparameter.
- Die Lösung der Aufgabe kann an den Aufrufer zurückgegeben werden.



Integrierte Funktionen

- Mathematische Funktionen für Winkelberechnungen etc.
- Bereitstellung von finanzmathematischen Funktionen.
- Datums- und Zeitwerte berechnen und umwandeln.
- Strings bearbeiten.
- Datentypen konvertieren und überprüfen.
- Datenfelder initialisieren und bearbeiten.
- Daten aus Dateien ein- und auslesen.
- Arbeiten mit Verzeichnissen und Dateien.
- ... haben eine englischsprachige Bezeichnung. Eine deutsch-englische Übersetzung für Office 2007 finden Sie unter http://www.piuha.fi/excel-function-name-translation/index.php?page=deutsch-english.html

Heute

```
Sub datumFunc()
    dim datumZeit As Date
    Dim datum As Date
    dim zeit As Date

    datumZeit = Now()    ' Aktuelles Datum und Zeit
    datum = Date         ' Aktuelles Systemdatum
    zeit = Time()        ' Aktuelle Systemzeit
End Sub
```

Datumswerte subtrahieren

```
Sub datumSub()
    Dim datum As Date
    Dim tage As Integer
    Dim monate As Integer

    datum = Date
    tage = DateDiff("d", datum, #7/31/2009#)
    monate = DateDiff("m", datum, #7/31/2009#)

End Sub
```

Hinweise

- Der dritte Parameter wird vom zweiten Parameter der Funktion subtrahiert. Die Datumswerte können auch als String angegeben werden.
- Der erste Parameter gibt die Einheit des Rückgabewertes an. Beispiel:
  - "m" gibt die Anzahl Monate zurück.
  - "d" gibt die Anzahl Tage zurück.
- Es wird immer ein Wert vom Datentyp Long zurückgegeben.

Datumswerte addieren

```
Sub datumAdd()
    Dim oldDatum As Date
    Dim newDatum As Date
    Dim tage As Integer
    Dim monate As Integer

    oldDatum = Date
    tage = 3
    monate = 3

    newDatum = DateAdd("m", monate, oldDatum)
    newDatum = DateAdd("d", tage, oldDatum)
End Sub
```

Hinweise

- Zu dem dritten Parameter wird eine bestimmte Anzahl von Einheiten addiert. Der zweite Parameter gibt die Anzahl an.
- Der erste Parameter gibt die Einheit des zu addierenden Wertes an. Beispiel:
  - "m" gibt die Anzahl Monate zurück.
  - "d" gibt die Anzahl Tage zurück.
- Als Ergebnis wird das errechnete Datum zurückgeliefert.

Tage, Monate etc. filtern

```
Sub datumAdd()
    Dim oldDatum As Date
    Dim newDatum As Date
    Dim tage As Integer
    Dim monate As Integer
    Dim jahr As Integer

    oldDatum = Date
    tage = 3
    monate = 3

    newDatum = DateAdd("m", monate, oldDatum)

    jahr = DatePart("yyyy", newDatum)
End Sub
```

Aus einem Datum wird der Tag, der Monat, das Jahr etc. herausgefiltert. Mit Hilfe des ersten Parameters wird die Einheit zum Filtern festgelegt.

String-Funktionen

- ... können Strings teilen.
- ... können die Position eines Zeichens ermitteln oder Zeichen ersetzen.
- ... liefern in Abhängigkeit der Bezeichnung einen Rückgabewert zurück.
  - Format() liefert eine Variable vom Datentyp Variant zurück.
  - Format$() liefert eine Variable vom Datentyp String zurück. Eine automatische Typ-Überprüfung findet statt.

Länge eines Strings

```
Sub WorkString()
    Const txtSatz As String = "Eisbären leben in der Arktis"
    Dim laenge As Integer
    Dim pos As Integer
    Dim tmpString As String

    laenge = Len("Eisbären")
    tmpString = Left(txtSatz, laenge)

    laenge = Len("Arktis")
    tmpString = Right(txtSatz, laenge)

    pos = InStr(txtSatz, "leben")
    laenge = Len("leben")
    tmpString = Mid(txtSatz, pos, laenge)
End Sub
```

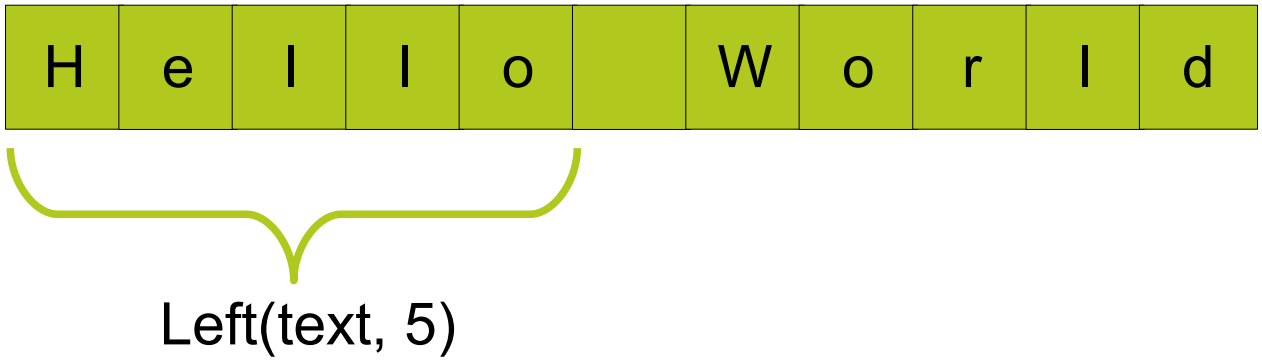
Die Länge eines Strings wird ermittelt. Die Länge eines leeren Strings ist gleich Null. Die Länge eines undefinierten Strings liefert ein Fehler. Mit Hilfe der Funktion Nz(string) kann ein undefinierter String in einen leeren String konvertiert werden.

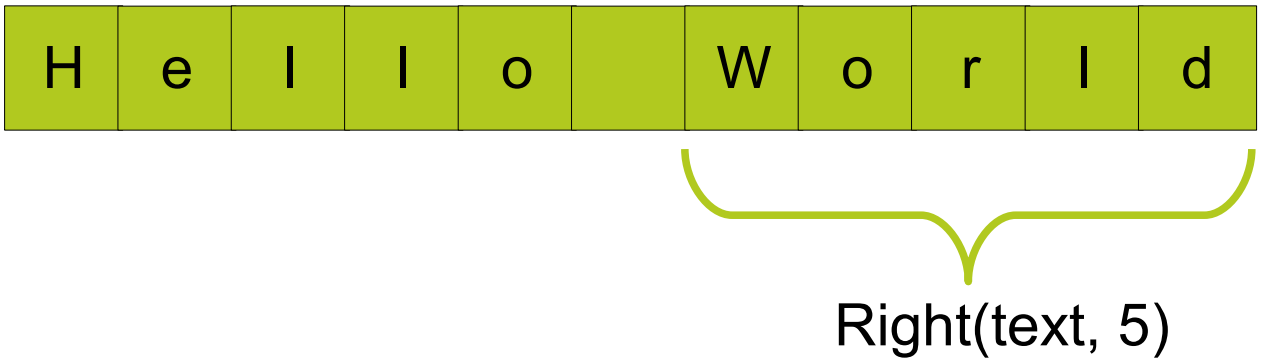
Teilstrings

```
Sub WorkString()
    Const txtSatz As String = "Eisbären leben in der Arktis"
    Dim laenge As Integer
    Dim pos As Integer
    Dim tmpString As String

    laenge = Len("Eisbären")
    tmpString = Left(txtSatz, laenge)

    laenge = Len("Arktis")
    tmpString = Right(txtSatz, laenge)

    pos = InStr(txtSatz, "leben")
    laenge = Len("leben")
    tmpString = Mid(txtSatz, pos, laenge)
End Sub
```

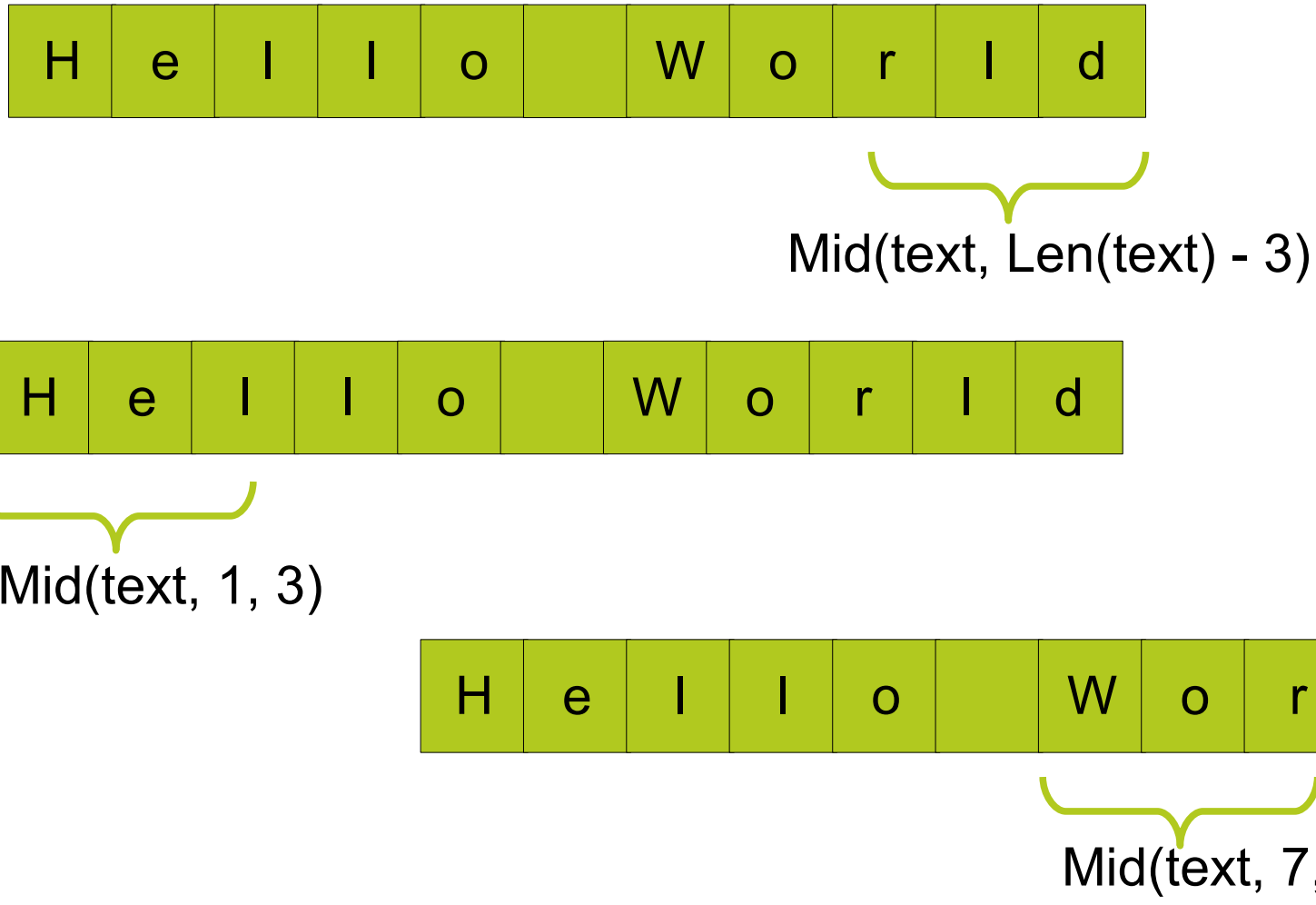
Mit Hilfe der Funktionen Left(String), Right(String) oder Mid(String) wird ein bestimmter Teil aus einem String gefiltert.

Arbeitsweise von Left() und Right()





Arbeitsweise von Mid()



Teilstrings suchen

```
Sub WorkString()
    Const txtSatz As String = "Eisbären leben in der Arktis"
    Dim laenge As Integer
    Dim pos As Integer
    Dim tmpString As String

    laenge = Len("Eisbären")
    tmpString = Left(txtSatz, laenge)

    laenge = Len("Arktis")
    tmpString = Right(txtSatz, laenge)

    pos = InStr(txtSatz, "leben")
    laenge = Len("leben")
    tmpString = Mid(txtSatz, pos, laenge)
End Sub
```

Mit Hilfe der Funktionen Instr(String, Suche) kann nach einem Zeichen oder einer Zeichenkette in einem anderen String gesucht werden. Die Position wird zurückgeliefert.

Hinweise

- Folgende Parameter sind in der angegebenen Reihenfolge möglich:
  - Ab welcher Position wird mit der Suche begonnen? Der Parameter ist optional.
  - Welcher String soll durchsucht werden?
  - Nach welchen String wird gesucht?
- Als Rückgabewert wird
  - ... ein Wert größer gleich eins geliefert, wenn die Suche erfolgreich war.
  - ... Null zurückgeliefert, wenn das Suchmuster nicht vorhanden ist.

... suchen und ersetzen

```
Sub ReplaceString()
    Const baeren As String = "Eisbären, Braunbären, Grizzlybären, Nasenbären"
    Const oldZeichen As String = "ä"
    Const newZeichen As String = "ae"
    Dim newText As String

    newText = Replace(baeren, oldZeichen, newZeichen)
    newText = Replace(baeren, oldZeichen, newZeichen, Len("Eisbären") + 2)

    newText = Replace(baeren, oldZeichen, newZeichen, , 2)

    newText = Replace(baeren, oldZeichen, newZeichen, , , vbTextCompare)

End Sub
```

Möglichkeiten

```
newText = Replace(baeren, oldZeichen, newZeichen)
```

- In dem angegebenen String (1. Parameter) werden alle Zeichen (2. Parameter) durch ein anderes Zeichen (3. Parameter) ersetzt.
- Es wird ein String mit den Ersetzungen zurückgeliefert.

```
newText = Replace(baeren, oldZeichen, newZeichen, _
                                    Len("Eisbären") + 2)
```

- Als vierter Parameter wird eine Startposition für die Suche angegeben.
- Dieser und alle nachfolgenden Parameter sind optional.

Möglichkeiten

```
newText = Replace(baeren, oldZeichen, newZeichen, , 2)
```

- Als fünfter Parameter wird die Anzahl der Ersetzungen angegeben.

```
newText = Replace(baeren, oldZeichen, newZeichen, , , _
                                    vbTextCompare)
```

- Als sechster Parameter kann eine Vergleichsart angegeben werden.
- In diesem Beispiel wird ein textlicher Vergleich durchgeführt. Die Groß- und Kleinschreibung wird nicht beachtet.

Leerzeichen entfernen

```
Sub ReplaceString()
    Const baeren As String = " Eisbären, Braunbären, Grizzlybären, Nasenbären "
    Dim newText As String

    newText = LTrim$(baeren)  ' Leerzeichen am Anfang werden entfernen

    newText = RTrim$(baeren)  ' Leerzeichen am Ende entfernen

    newText = Trim$(baeren)    ' Leerzeichen am Anfang und Ende entfernen


End Sub
```

Formatierungen für Datums- und Zeitangaben

```
Dim  text As String
Dim tmpText As String

text = Date
tmpText = Format(text, "Long Date")          ' Dienstag, 9. April 2007

text = Time
tmpText = Format(text, "Short Time")         ' 09:37

text = #12/07/2007#
tmpText = Format(text, "dd.mmmm.yyyy") ' 07. Dezember. 2007
```

Formatierungen für Ganz- und Dezimalzahlen

```
Dim  text As String
Dim tmpText As String

text = "12345,64454"
tmpText = Format(text, "Currency")          ' 12.345,64 €

text = 1230.3
tmpText = Format(text, "##,##0.00")         ' 1.230.30

text = 0
tmpText = Format(text, "Yes/No")            ' Nein
```

Hinweise

- Der Funktion wird als erster Parameter der zu formatierende String übergeben.
- Es können vorgefertigte oder mit Hilfe von Formatierungszeichen definierte Formatstrings genutzt werden. In der Hilfe finden Sie unter folgenden Stichwörtern Formatierungszeichen:
  - Benutzerdefinierte Datums- und Zeitformate.
  - Benutzerdefinierte numerische Formate.
  - Benutzerdefinierte Zeichenfolgenformate.
- Der Wert der Variablen wird nicht verändert. Die Darstellung des Wertes wird verändert.

Beispiele für nummerische Formate

- Format(text, "##,##0.00")
  - Die Null sowie das Hash-Zeichen sind Formatierungszeichen für eine Ziffer an dieser Position.
  - Das Hash-Zeichen zeigt keine Zahl an, wenn keine Ziffer an der Position vorhanden ist.
  - Die Null zeigt eine Null an, wenn keine Zahl vorhanden ist.
  - Das Komma ist ein Platzhalter für das Tausender-Trennzeichen
  - Der Punkt ist ein Platzhalter für das Dezimaltrennzeichen.

Beispiele für Datumsangaben.

- Format(text, "dd.mmmm.yyyy")
  - dd formatiert Tagesangaben zweistellig
  - mmmm schreibt den Monat vollständig aus.
  - yyyy zeigt das Jahr vierstellig an.

Zellen formatieren

```
Sub FuncWorkSheet()

    Range("A13").NumberFormatLocal = "0.0 ""Grad C"""
    Range("B13").NumberFormatLocal = "0.0 ""Grad F"""

End Sub
```

Die Zellen werden in der lokalen Sprache des Benutzers formatiert.
Die doppelten Anführungsstriche stellen ein Anführungsstrich in einer Zeichenkette dar. Die Funktion Format und NumberFormatLocal nutzen die gleichen Formatierungszeichen.

Funktionen aus Excel nutzen

```
Sub FuncWorkSheet()
    Dim wert As Integer

    wert = Range("A13").Value
    Range("B13").Value =
                Application.WorksheetFunction.Convert(wert, "C", "F")

End Sub
```

Application.WorksheetFunction enthalten Tabellenfunktionen aus Excel. Hier wird die Funktion Convert für eine Konvertierung von Celsius-Werten in Fahrenheiten-Werten genutzt.

Zellen zählen

```
Dim anzahl As Integer

anzahl = Application.WorksheetFunction.Count(Range("C5:D10"))

anzahl = Application.WorksheetFunction.CountBlank(Range("C5:D10"))
```

Die Funktion Count zählt Zellen in einem Bereich, die Zahlen oder Datumsangaben enthalten.
Die Funktion CountBlank zählt leere Zellen in einem bestimmten Zellbereich.

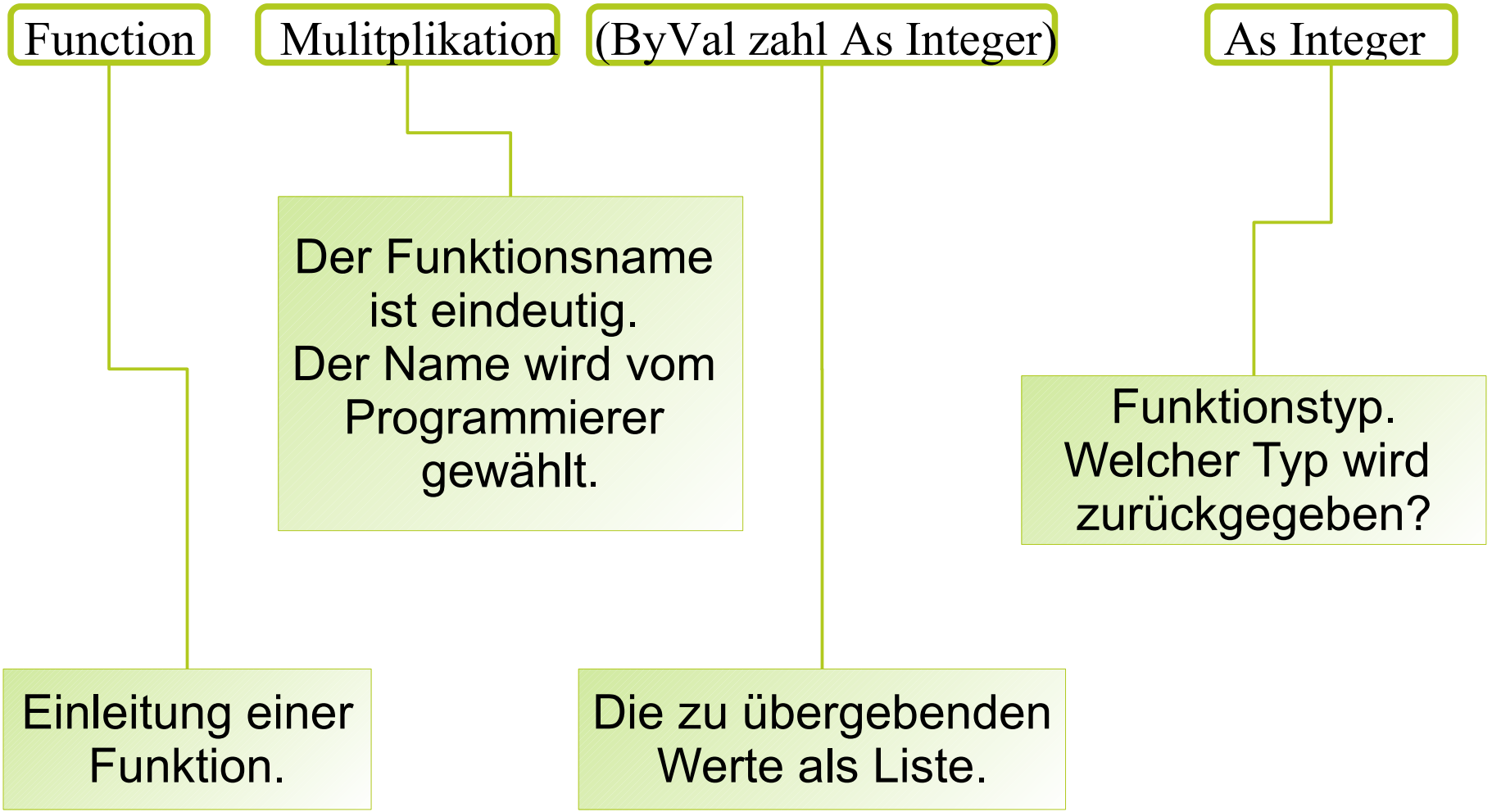
Funktionen in VBA erstellen

```
Function Mulitplikation(ByVal zahl As Integer) As Integer
    Dim ergebnis As Integer

    ergebnis = zahl * zahl

    Multiplikation = ergebnis
End Function
```

Funktionen bestehen aus

- ... dem Funktionskopf Function Funkt (para01, par02) As Datentyp.
  - Schnittstelle nach außen.
  - Wie wird die Funktion aufgerufen?
  - Art des Rückgabewertes. Funktionstyp.
- ... dem Funktionsrumpf.
  - Auszuführende Anweisungen.
  - Eine Anweisung, die einen Wert an den Aufrufer zurückgibt.
- Jede Prozedur endet mit End Function.

Funktionskopf



Funktionstyp

- Dem Funktionsnamen wird mit Hilfe des Schlüsselwortes As ein Datentyp zugeordnet.
- Der Funktionstyp entspricht dem Datentyp des Rückgabewertes der Funktion.
- Als Funktionstyp kann jeder beliebiger Datentyp genutzt werden.
- Falls kein Datentyp angegeben ist, wird der Datentyp des Rückgabewertes genutzt. Nachteil: Der Aufrufer der Funktion weiß nicht, welche Art von Wert zurückgegeben wird.

Rückgabewert

Multiplikation = ergebnis

- Jede Funktion kann exakt einen Wert an den Aufrufer zurückgeben.
- ... hat den gleichen Datentyp wie die Funktion.
- ... kann in der Funktion berechnet werden.
- ... kann an jeder beliebigen Stelle in einer Funktion auftreten. Häufig steht die Anweisung am Ende einer Funktion.
- ... kann in Abhängigkeit einer Bedingung zurückgegeben werden.
- Mit Hilfe der Anweisung wird die Funktion nicht verlassen. Die Funktion wird durch die Anweisung End Function beendet. Mit Hilfe von Exit Function kann eine Funktion vorzeitig verlassen werden.

Möglichkeiten für den Funktionsaufruf

ergebnis = Multiplikation(zahl)

- Die Funktion wird genauso wie eine Prozedur mit ihren Namen aufgerufen.
- Dem Namen folgt die Parameterliste in Klammern.
- Der Rückgabewert der Funktion wird mit Hilfe des Gleichheitszeichens einer Variablen zugewiesen.

Call Multiplikation(zahl)

- Der Rückgabewert der Funktion wird nicht verarbeitet.
- Ein Fehler wird nicht angezeigt.

ergebnis = Berechnung(Multiplikation(zahl), zahl)

- Das Ergebnis der Funktion wird direkt weitergereicht.

Module

- ... fassen Prozeduren, die eine gemeinsame Aufgabe bearbeiten, zusammen.
- ... kapseln Code zu einem bestimmten Thema.
- ... sind die Kapitel in einem Buch. Die Funktionen in einem Modul symbolisieren die verschiedenen Abschnitte.
- ... sind eine Sammlung von Deklarationen, Anweisungen und Prozeduren, die im Rahmen eines bestimmten Projekts gespeichert werden.
- ... ermöglichen eine strukturierte Programmierung.
- In einem Projekt kann mehr als ein Modul vorhanden sein.

Mit Modulen arbeiten

- *Datei – Datei importieren* fügt dem aktiven Projekt ein neues Modul aus einem anderen Projekt hinzu.
- *Datei – Datei exportieren* speichert das Modul in einer Datei.
- Im Eigenschaften-Fenster kann der Name eines Moduls geändert werden.
- Module werden mit dem Projekt gespeichert.

Standardmodule

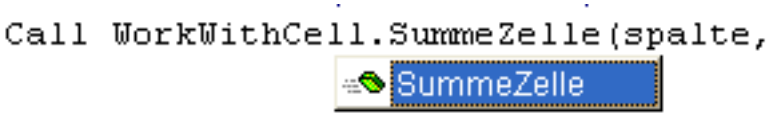
- ... sind nicht an ein bestimmtes Objekt gebunden.
- ... enthalten häufig Code, der innerhalb des gesamten Projekts von vielen Elementen genutzt wird.
- ... bekommen die Dateiendung ".bas".
- Mit Hilfe von *Einfügen* – *Modul* wird dem aktiven Projekt ein neues Modul hinzugefügt.

Zugriffsrechte von Funktion und Prozeduren

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL As Integer) As Integer
    nSumme = zahlLinks + zahlRechts
    Return nSumme
End Function

Sub Main()
        Dim nZahl01 As Integer
        Dim nZahl02 As Integer

        nSumme = add(nZahl01, nZahl02)
End Sub
```

Standardmäßig sind alle Prozeduren und Funktionen in einem Modul öffentlich. Das heißt jeder kann auf die Unterprogramme zugreifen.

Unterprogramme in anderen Modulen



- Voraussetzung: Die Prozedur ist nicht mit Private Sub oder Private Function gekennzeichnet.
- Geben Sie den Modulnamen, gefolgt von einem Punkt ein. Im Anschluss daran geben Sie den Prozedurnamen an.
- Falls Sie für den Editor die Option Elemente automatisch auflisten aktiviert haben, werden alle öffentlichen Prozeduren eines Moduls in einer Liste angezeigt.

Klassenmodule

- ... fassen Objekte in Kategorien zusammen.
- ... fassen Objekte mit den gleichen Eigenschaften und Methoden zusammen.
- ... stellen Baupläne für konkrete Objekte dar.
- ... bieten eine Schablone für bestimmte Kategorien von Objekten.
- ... bekommen die Dateiendung ".cls".

Klassenmodule erstellen

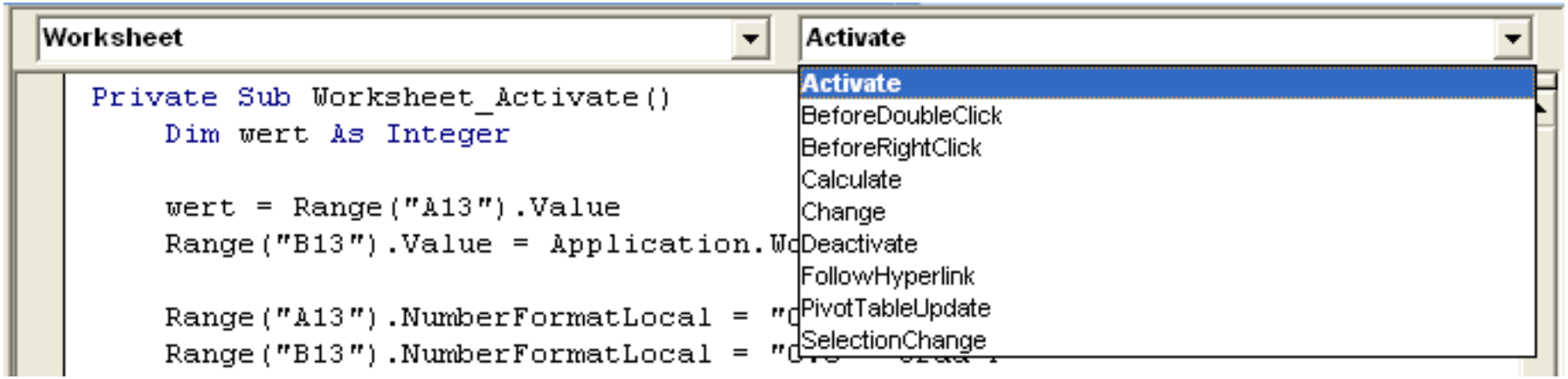
- Klassenmodule werden mit einem Tabellenblatt oder einer Arbeitsmappe automatisch erstellt.
- Mit Hilfe von *Einfügen* – *Klassenmodul* wird dem aktiven Projekt ein neues Klassenmodul hinzugefügt. Die Erstellung von eigenen Klassen ist nicht Bestandteil dieses Kurses.

Ereignisprozeduren

... werden durch eine Aktion eines Benutzer ausgelöst. Diese Prozedur wird zum Beispiel durch die Aktivierung des Tabellenblattes aufgerufen.
Diese Art von Prozeduren können nicht im Code aufgerufen werden.

```
Private Sub Worksheet_Activate()
    Dim wert As Integer
    Range("A13").NumberFormatLocal = "0.0 ""Grad C"""
    Range("B13").NumberFormatLocal = "0.0 ""Grad F"""
End Sub
```

Objekte und Prozeduren im Codefenster



- Im ersten Kombinationsfeld wird ein Objekt ausgewählt. Das Objekt Allgemein ist immer vorhanden. Das Objekt listet alle Prozeduren und Funktionen auf, die keinem Objekt zugeordnet werden können.
- In Abhängigkeit des Objekts können in dem zweiten Kombinationsfeld Ereignisse oder die dazugehörigen Prozeduren ausgewählt werden. Der Eintrag Deklaration enthält Definition, die das gesamte Modul betreffen.

Zugriffsrechte von Funktion und Prozeduren

In einem Klassenmodul sind Funktionen generell privat. Das heißt, sie sind nicht von außen zugänglich.

```
Private Sub Worksheet_Activate()
    Dim wert As Integer
    Range("A13").NumberFormatLocal = "0.0 ""Grad C"""
    Range("B13").NumberFormatLocal = "0.0 ""Grad F"""
End Sub
```

Variablen und Blöcke

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL As Integer) As Integer
    nSumme = zahlLinks + zahlRechts
    Return nSumme
End Function

Sub Main()
        Dim nZahl01 As Integer
        Dim nZahl02 As Integer

        nSumme = add(nZahl01, nZahl02)
End Sub
```

In jedem Block gibt es jede Variable nur einmal. Der Variablenname ist eindeutig.

Gültigkeit

- Die Variable ist innerhalb der Prozedur oder Funktion definiert. Die Variable wird als Parameter genutzt.
  - Lokale Variablen.
  - Die Variablen sind nur innerhalb der Prozedur oder Funktion gültig.
- Die Variable ist am Anfang des Moduls definiert.
  - Globale Variablen.
  - Die Variablen sind im gesamten Modul gültig.
  - Globale Variablen sollten nur in Ausnahmefällen genutzt werden.
- Nach Verlassen des Blocks wird die Variable zerstört.

Lokale Variablen

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL As Integer) As Integer
      nSumme = zahlLinks + zahlRechts
      Return nSumme
End Function

Sub Main()
            Dim nZahl01 As Integer
            Dim nZahl02 As Integer

            nSumme = add(nZahl01, nZahl02)
End Sub
```

Diese Variablen sind nur zwischen den Anweisungen Sub oder Function und End ... bekannt. Die Variablen können innerhalb dieses Blockes genutzt werden.

Parameter einer Funktion

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL As Integer) As Integer
    nSumme = zahlLinks + zahlRechts
    Return nSumme
End Function

Sub Main()
        Dim nZahl01 As Integer
        Dim nZahl02 As Integer

        nSumme = add(nZahl01, nZahl02)
End Sub
```

Parameter sind immer lokal.

Globale Variablen

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL As Integer) As Integer
    nSumme = zahlL + zahlRechts
    Return nSumme
End Function

Sub Main()
    Dim nZahl01 As Integer
    Dim nZahl02 As Integer

    nSumme = add(nZahl01, nZahl02)
End Sub
```

Diese Variable ist im gesamten Module bekannt. Die Variable kann auch innerhalb der Funktionen und Prozeduren in diesem Modul genutzt werden. Die Variable wird nach dem Verlassen des Moduls zerstört.

Sichtbarkeit

```
Dim nSumme As Integer

Function add(zahlR As Integer, zahlL A
    nSumme = zahlLinks + zahlRechts
    Return nSumme
End Function

Sub Main()
        Dim nZahl01 As Integer
        Dim nZahl02 As Integer
        Dim nSumme As Integer

        nSumme = add(nZahl01, nZahl02)
End Sub
```

Die lokale Variable nSumme überlagert die globale Variable nSumme. Die globale Variable ist in der Prozedur nicht sichtbar, aber gültig.

Zugriffmodifizierer für Variablen

- Private oder Dim
  - Die Variable ist nur lokal gültig.
  - Sie ist in ihren Block gekapselt.
  - Innerhalb einer Prozedur werden Variablen mit Dim deklariert.
- Public
  - Die Variablen sind öffentlich.
  - Jeder kann auf die Variable zugreifen.
  - ... darf nur auf Modulebene genutzt werden.
  - Die Variable kann im gesamten Projekt genutzt werden.

Statische Variablen

```
Sub plusEins()
   Static nSumme As Integer

   nSumme = nSumme + 1
   Debug.Print nSumme
End Sub


Sub Main()
   plusEins
   plusEins
   plusEins
End Sub
```

Diese Variable wird nicht am Ende des Blocks zerstört. Ihre Lebensdauer wird über das Ende des dazugehörigen Blocks verlängert. Statische Variablen können nicht innerhalb eines Moduls definiert werden.