

Prozeduren und Funktionen

Parameter sind Platzhalter in einer Prozedur oder Funktion (Methode), die erst bei einem konkreten Aufruf der Methode festgelegt werden. Die in einer Methode stehenden Platzhalter bezeichnet man als **formale Parameter**, die im Aufruf der Methode stehenden Werte als **aktuelle Parameter**.

Aufbau einer Prozedur

Jede Prozedur besteht aus dem Schlüsselwort `procedure`, gefolgt von einem gültigen Namen und eventuell einer Parameterliste in runden Klammern. Sind keine Parameter vorhanden, werden die Klammern sowohl bei der Deklaration als auch beim Aufruf weggelassen. Diesen Teil nennt man Kopf der Prozedur. Es folgen **Variablen-** und **Konstantendeklarationen** und anschließend zwischen `begin` und `end` die Anweisungen, die die Prozedur durchführen soll:

```
procedure <Name>( <Parameter> );  
    <(lokale)Variablen und Konstanten>  
begin  
    <Anweisungen>  
end;
```

Beispiel:

Die folgende Methode gibt so viele Töne über den PC-Lautsprecher aus, wie über den Parameter „anzahl“ angegeben.

```
procedure ToeneAusgeben(anzahl: integer);  
    var i : integer;  
begin  
    for i := 1 to anzahl do  
        beep;  
    end;
```



Wie lautet der Aufruf der Methode „ToeneAusgeben“ für 10 Töne?

Der Aufruf der Methode für 10 Töne geschieht folgendermaßen:

Aufbau einer Funktion

Eine Funktion unterscheidet sich nur geringfügig von einer Prozedur. Sie besitzt einen Rückgabewert und wird mit dem Schlüsselwort `function` deklariert an Stelle von `procedure`.

```
function <Name>(<Parameter>): <Rückgabotyp>;
  <(lokale)Variablen und Konstanten>
  begin
    <Anweisungen>
  end;
```

Der Funktionsname ist ein beliebiger gültiger Bezeichner; der Rückgabotyp ein beliebiger Datentyp (wie z. B. `integer` oder `real`).

Beispiel:

Eine Funktion, die drei Zahlen addiert und das Ergebnis zurückliefert.

```
...
function SummeAusDrei(zahl1, zahl2, zahl3 : integer): integer;
  begin
    result := zahl1 + zahl2 + zahl3;
  end;
...
```

Bei `result` handelt es sich um eine vordefinierte Variable, der den Rückgabewert der Funktion enthält. Der Rückgabewert kann dann an der Aufrufstelle ausgewertet werden:

```
...
procedure TMyMax.BerechnenBtnClick(Sender: TObject);
  var ergebnis : integer;
  begin
    ergebnis := SummeAusDrei(3,5,9); //Aufruf der Funktion
  end;
...
```

Funktion oder Prozedur?

Die Entscheidung, ob man eine Methode nun besser in eine Prozedur verpacken sollte, bleibt letztlich Ihnen überlassen. Prinzipiell sind beide Wege gangbar, da nicht nur Funktionen, sondern auch Prozeduren Werte zurückliefern können. Allerdings gilt folgende Empfehlung:

Ist man nur an einem einzigen Rückgabeparameter interessiert, so sollte man dafür eine Funktion schreiben, bei mehreren Rückgabeparametern sind aber Prozeduren vorzuziehen (*siehe hierzu auch S. 5*).

Bei Funktionen ist außerdem zu beachten, dass bei der Deklaration ein Datentyp zuzuordnen ist.

Parameterübergabe

Beim Aufruf einer Methode mit Parametern muss beachtet werden, dass Anzahl und Typ der Werte übereinstimmen. Anhand der Reihenfolge der Werte steht in obigem Beispiel (S. 2) fest, dass die Variable `zahl1` den Wert 3, `zahl2` den Wert 5 und `zahl3` den Wert 9 erhält. Diese Variablen werden nicht wie üblich über `var` deklariert. Ihre Deklaration erfolgt durch die Nennung im Funktionskopf. Außerdem gelten sie nur innerhalb der Funktion (**lokale Variable**). Von außerhalb (z. B. nach Beendigung der Funktion) kann nicht mehr auf sie zugegriffen werden.

Werte- und Variablenparameter

Wird eine Methode aufgerufen, so erfolgt die Parameterübergabe normalerweise als Wert (**call by value** → *Werteparameter*). Da man hier mit einer Kopie des Originalwertes arbeitet, ist es im Körper der Routine unmöglich, am Original etwas zu ändern. Man kann das aber durch Voranstellen des reservierten Wortes `var` erreichen. In diesem Fall erfolgt die Datenübergabe als Referenz (**call by reference** → *Variablenparameter, manchmal auch Referenzparameter genannt*). Dabei wird intern auf das Original, also auf die Adresse der Variablen verwiesen. Der aufgerufenen Methode ist es nunmehr möglich, z. B. das Ergebnis einer Berechnung in dieser Variablen zurückzuliefern (*siehe hierzu auch Abbildung auf S. 6*).

1. Werteparameter

In den obigen Beispielen wird beispielsweise immer eine Kopie eines Wertes an die Methode übergeben. Wenn dieser Wert also innerhalb der Prozedur bzw. Funktion verändert wird, ändert sich nicht die Variable, die beim Aufruf verwendet wurde:

```

procedure MachWas(zahl: integer);
  begin
    zahl := zahl + 5;
  end;
...
procedure Aufrufen;
var IrgendEineZahl: integer;
begin
  IrgendEineZahl := 5;
  MachWas(IrgendEineZahl);
end;

```

Im Beispiel ruft die Prozedur Aufrufen die Prozedur MachWas mit dem Wert der Variable IrgendEineZahl auf. In MachWas wird dieser Wert über zahl angesprochen. Und obwohl zahl nun verändert wird, ändert sich der Wert der Variablen IrgendEineZahl nicht. Er ist am Ende immer noch 5. Man spricht hier von einem **Werteparameter**, da nur der Inhalt der Variablen übergeben wird.

Als Werteparameter können auch Ausdrücke übergeben werden, was bei Variablenparametern nicht möglich ist. Solche Ausdrücke werden vor ihrer Übergabe automatisch ausgewertet und das Ergebnis der Auswertung anschließend als aktueller Parameter an die aufgerufene Prozedur oder Funktion übergeben.

Beispiel:

```
Bruttopreis := RundeKfm(12.67 / 100 * 15);
```

2. Variablenparameter

Im Fall des **Variablenparameters** wird das „Original“ übergeben. Ein solcher Parameter wird mit dem Schlüsselwort **var** gekennzeichnet.

```

procedure MachWas(var zahl: integer);
  begin
    zahl := zahl + 5;
  end;
...
procedure Aufrufen;
var IrgendEineZahl : integer;
begin
  IrgendEineZahl := 5;
  MachWas(IrgendEineZahl);
end;

```

Hier wird keine Kopie des Variableninhalts übergeben, sondern eine Referenz (also die Speicheradresse) der Variablen `IrgendEineZahl`. Wird der Wert in `MachWas` nun um 5 erhöht, geschieht dies auch mit der Variablen `IrgendEineZahl`, weil es sich um dieselbe Variable im Speicher handelt. Sie wird nur von den beiden Prozeduren mit anderen Namen angesprochen. Im Gegensatz zum Fall der Wertparameter braucht ein Variablenparameter in einer Prozedur also keinen zusätzlichen Speicherplatz, da die „originale Variable“ weiterverwendet wird.

Über den Umweg des „var-Parameters“ kann man sogar Prozeduren dazu bewegen, Werte zurückzugeben:

```

procedure TuWas(var anzahl1, anzahl2: integer);
  begin
    anzahl1 := 9;
    anzahl2 := 8;
  end;
...
procedure Aufrufen;
var zahl1, zahl2: integer;
begin
  TuWas(zahl1, zahl2);
end;

```

In `TuWas` werden `anzahl1` und `anzahl2` Werte zugewiesen - und da es sich dabei um Variablenparameter handelt, automatisch auch `zahl1` und `zahl2`. Wenn `TuWas` abgearbeitet wurde, enthält

`zahl1` den Wert und

`zahl2` den Wert

3. Konstantenparameter

Im Gegensatz zu Variablenparametern ist ein nur lesender Zugriff auf den aktuellen Parameter möglich. (Der aktuelle Parameter ist schreibgeschützt.) Konstantenparameter müssen in der Prozedurschnittstelle mit `const` deklariert werden:

```

procedure MachWas(const zahl: integer);

```

Welche Art von Parameter wann einsetzen?

- Wenn es die Aufgabe der Methode ist, den Parameter zu verändern, dann verwende **var**.
- Wenn ein Parameter viel Speicherplatz belegt, dann verwende **var**.
- Wenn keine dieser Bedingungen zutrifft, ist es meistens am besten, man verwendet **var** nicht.
- Wenn es zweckmäßig ist, der Methode einen Ausdruck zu übergeben, dann verwende **var** nicht.

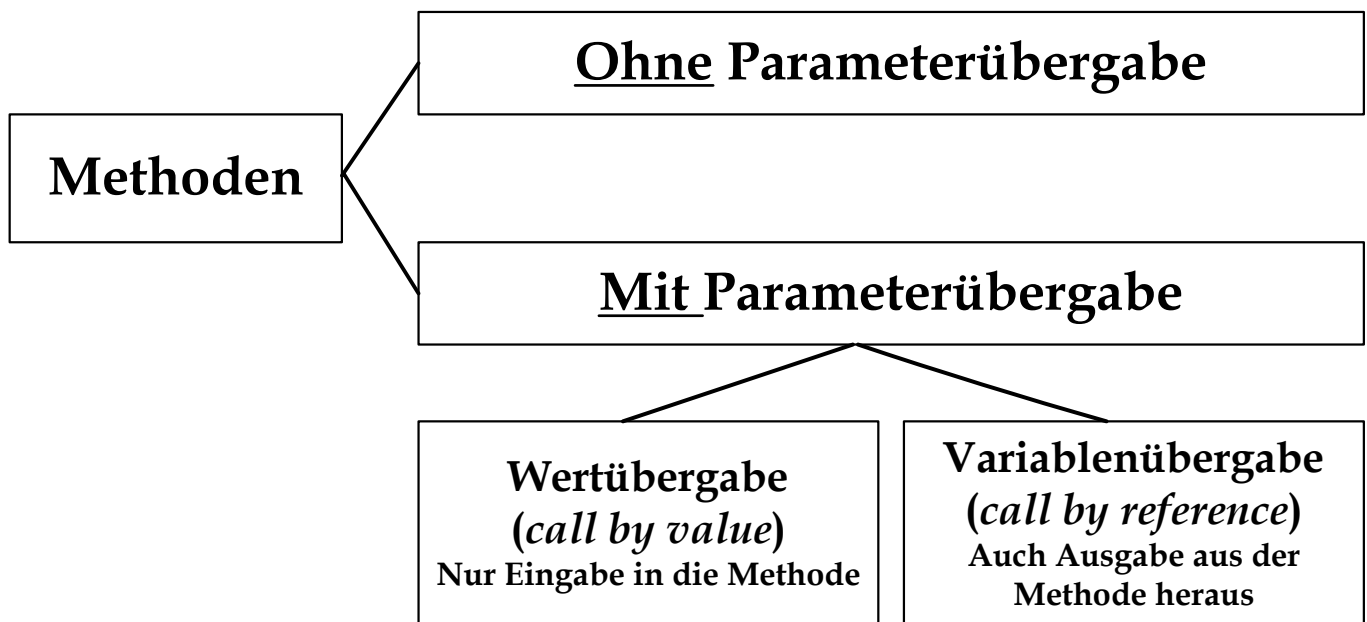


Abb.: Wert- und Variablenparameter