Der Oberbegriff „Funktion“ umfasst in der Informatik zwei unterschiedliche Konzepte:

* Funktionen
* Prozeduren

Der Unterschied liegt hierbei darin, wie das jeweilige Konstrukt innerhalb des Codes verwendet wird.

Eine Funktion gibt einen Rückgabewert aus. Eine Prozedur tut etwas.

Der Übergang zwischen den beiden ist fließend, da oft Prozeduren Rückgabewerte haben und Funktionen über sogenannte „Nebenwirkungen“ verfügen. In diesem Text ist „Funktion“ ein Synonym für beide Begriffe, aber auch ein Begriff für die reine Funktion.

Beispiel:

void SchreibeZahl (int Zahl);

int max (int Nummer1, int Nummer2);

Der Unterschied ist leicht auszumachen. Bei der Funktion steht „int“ als Rückgabewert, bei der Prozedur „void“. Natürlich kann neben int auch jeder andere Datentyp als Rückgabewert genutzt werden. Void steht für „kein Rückgabewert“ und damit immer eine Prozedur.

Der Aufbau einer Funktionsdeklaration, wie sie gerade als Beispiel genutzt wurde, sieht so aus:

[Rückgabewert] [Funktionsname] (

[Parameter 1 Datentyp] [Parameter 1 Name],

[Parameter 2…],

[etc.]

);

Parameter sind Werte, die an eine Funktion übergeben werden. Der Rückgabewert ist etwas, das von einer Funktion nach ihrer Vollendung ausgeworfen wird.

Eine Funktionsdeklaration beschreibt die Form einer Funktion, ohne zu erklären, was diese eigentlich tut. Dafür wäre eine Funktionsdefinition notwendig.

Um eine Funktion zu nutzen, ruft man sie auf. Ein Funktionsaufruf kann unterschiedlich aussehen:

Int a = max (5, 7); //max() errechnet das Maximum. 7 ist höher als 5. Entsprechend ist der Rückgabewert 7.

Also steht dort: int a = 7;

SchreibeZahl (5); //Die Zahl „5“ erscheint irgendwo, z.B. in der Kommandozeile

Printf(„%d“, max(a, 12); //printf gibt einen Integer aus („%d“), dieser Integer ist das Maximum zwischen a aus dem ersten Beispiel (also 7) und 12

Int x = printf(„%d“, printf(„Hello World!“));

Das letzte Beispiel zeigt sehr gut, dass etwas, das wie eine Prozedur aussieht, eine Funktion sein kann. In der Standardbibliothek ist printf so beschrieben:

int printf (const char \*format, ...);

Man merke, dass hier ein Rückgabewert vom Typ Integer definiert ist. Sucht man im Netz, findet man Folgendes:

## Return Value

On success, the total number of characters written is returned.  
  
If a writing error occurs, the *error indicator* ([ferror](http://www.cplusplus.com/ferror)) is set and a negative number is returned.  
  
If a multibyte character encoding error occurs while writing wide characters, [errno](http://www.cplusplus.com/errno) is set to EILSEQ and a negative number is returned.

(http://www.cplusplus.com/reference/cstdio/printf/)

Kurz, printf hat als Rückgabewert die Anzahl der ausgegebenen Zeichen. Tritt ein Fehler auf, so ist der Rückgabewert -1.

Sieht man sich unser Beispiel nun an, finden dort drei Anweisungen statt:

Int x = printf(„%d“, printf(„Hello World!“));

1. Int x = [Ausdruck]
2. Printf(“%d”, [Ausdruck])
3. Printf(“Hello World”)

Derartige Ausdrücke werden von innen nach außen abgearbeitet.

Zuerst wird also „Hello World“ in der Konsole ausgegeben. Anschließend erscheint der Rückgabewert „12“ dort, wo vorher die printf-Anweisung stand, da „Hello World“ zwölf Zeichen lang ist.

Der zweite Ausdruck sieht danach also so aus:

Printf(„%d“, 12)

Entsprechend taucht in der Konsole die 12 als Ausgabe auf. Da weder Leerzeichen noch ein Zeilenumbruch nach dem Hello World folgen, sieht die Konsolenausgabe dann so aus:

Hello World12

12 hat zwei Zeichen. Also ist der Rückgabewert hier 2.

Int x = 2;

Drei Anweisungen, geschachtelt in einer Zeile.

Nebenwirkungen:

Wenn wir die Funktion max() nehmen, die weiter oben erwähnt wurde, so würden wir erwarten, dass diese einfach zwei Parameter nimmt und davon den höheren ausgibt.

Was aber wäre, wenn die Funktion so definiert wäre:

Int max (int Nummer1, int Nummer2) {

Printf(„Hallo! Ich bin die Max-Funktion!“);

If (Nummer1>Nummer2) {

return Nummer1;

} else {

return Nummer2

};

}

Die Nebenwirkung dieser Funktion wäre, dass sie Text auf der Konsole ausgibt und das auch noch in Abhängigkeit von einer anderen Funktion.

Dies könnte beispielsweise dazu führen, dass man ein Programm ohne Konsolenausgabe hat. In diesem Programm würde man also „stdio.h“ nicht inkludieren. Verwendet man nun max(), so gibt es einen Compiler-Fehler, weil max() eine Abhängigkeit zu stdio.h hat, die man eventuell garnicht erkannt hat.

Auch würde man sich möglicherweise wundern, wo der zusätzliche Text in der Konsole herkommt, wenn die Kompilierung doch klappt.

Deshalb sollte eine Funktion immer nur Parameter annehmen und Ergebnisse ausgeben und niemals Zwischenschritte haben, die Dinge außerhalb der Funktion beeinflussen.

Genauso sollten Prozeduren nur ihre Aufgabe erfüllen und nicht zur eierlegenden Wollmilchsau werden, die gleichzeitig das Bild grün macht und Pi als Rückgabewert ausgibt.

Hat eine Prozedur einen Rückgabewert, so sollte dieser ein Feedback über die Funktion der Prozedur geben. Printf ist da ein gutes Beispiel. Scanf ist ähnlich, da es die Anzahl an erfolgreichen Zuweisungen zurückgibt – im Fehlerfall also eine Null.