12.Vorlesung Grundlagen der Informatik

Dr. Christian Baun

Hochschule Darmstadt
Fachbereich Informatik
christian.baun@h-da.de

12.1.2012

Wiederholung vom letzten Mal

- Anwendungsschicht
  - Anwendungsprotokolle
    - Fernsteuerung von Computern mit Telnet
    - Übertragung von Daten mit dem Hypertext Transfer Protocol (HTTP)
- Informationen in Hypertext-Systemen mit Auszeichnungssprachen darstellen
  - Hypertext Markup Language (HTML)
  - Inhalt und Layout der Informationen voneinander trennen
    - Extensible Markup Language (XML)
    - Transformation von XML-Dokumenten mit XSLT

Heute

- Algorithmen
- Sprachen
- Weg vom Programm zum Maschinenprogramm
- Programmierparadigmen und Programmiersprachen
- Einführung in PHP
- Datentypen
- Kontrollstrukturen

Die Programmbeispiele verwenden die Programmiersprache PHP

- Warum PHP?
  - Einfache Syntax und gut lesbarer Quelltext
  - Umfangreiche Dokumentation online verfügbar
    - z.B. `http://www.php.net` und `http://www.selfphp.de`
  - PHP ist Teil der Vorlesung Webskripting im nächsten Semester

Algorithmus

- Anleitung (**Verfahren**) zur Lösung eines Problems
- Vergleichbar einem Kochrezept, Strickmuster oder einer Reparatur- oder Montageanweisung
- Datenstrukturen und Anweisungsfolgen sind im Algorithmus festgelegt
- Kochrezept, Strickmuster oder Reparatur-/Montageanweisungen werden von Menschen gelesen und interpretiert
- Im Gegensatz dazu werden Algorithmen von Maschinen ausgeführt
- Daraus folgt: ein Programm muss so geschrieben sein, dass der Computer es ausführen kann, ohne dass er es inhaltlich versteht

Eigenschaften von Algorithmen

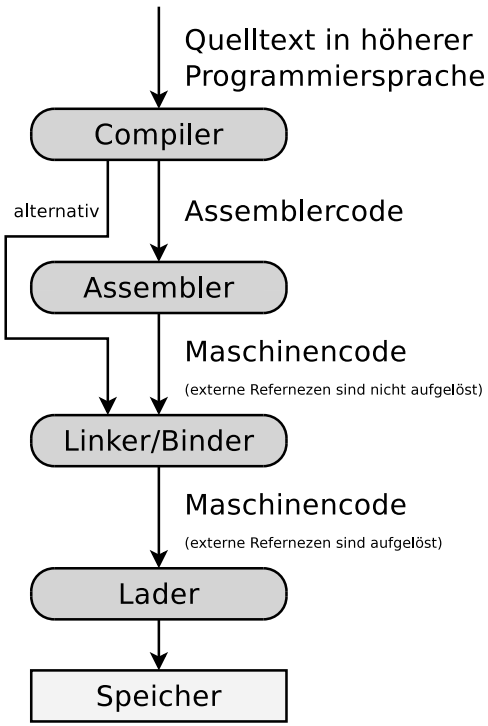
- **Ausführbarkeit**: Jeder Schritt muss tatsächlich ausführbar sein
- **Determiniertheit**: Gleiche Voraussetzungen (Eingabewerte) liefern immer das gleiche Ergebnis
- **Determinismus**: Der nächste Schritt ist zu jedem Zeitpunkt eindeutig definiert
- **Effektivität**: Effekt jeder Anweisung eines Algorithmus muss eindeutig festgelegt sein
- **Finitheit**: Ein Algorithmus muss in einem endlichen Text eindeutig beschreibbar sein
  - Quelltext muss aus einer begrenzten Anzahl von Zeilen bestehen
- **Platzkomplexität**: Ausführung darf nur begrenzt viel Speicherplatz benötigen
- **Zeitkomplexität** und **Terminierung**: Nur endlich viele Schritte dürfen nötig sein

Sprache

- Bei allen Sprachen müssen die Regeln der **Syntax** (Grammatik) und der **Semantik** (Bedeutung der Konstrukte) eingehalten werden
- Beispiel: Syntax und Semantik bei HTML/XML
- **Syntax**
  - Die meisten Tags haben einen Beginn- und End-Auszeichner (Tag)
    - HTML: End-Tag ist in einigen Fällen optional (darf man weglassen)
    - XML: End-Tag muss immer angegeben werden
  - Groß- und Kleinschreibung der Tags
    - HTML: Groß- und Kleinschreibung wird ignoriert
    - XML: Groß- und Kleinschreibung der Tags wird nicht ignoriert
- **Semantik**
  - Bedeutung der Auszeichner
    - HTML: Die Bedeutung der Auszeichner ist festgelegt
    - XML: Die Auszeichner und damit auch ihre Bedeutung sind in der Document Type Definition (DTD) frei wählbar

Weg vom Programm zum Maschinenprogramm (1/3)

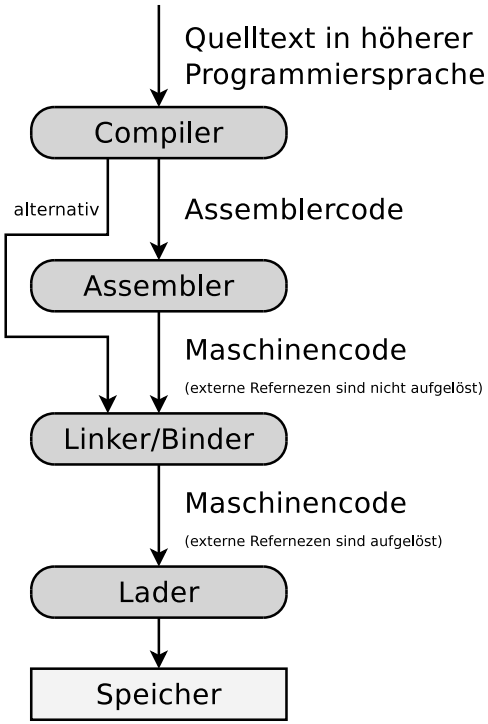
- Zwischen dem Quelltext und dem lauffähigen Programm gibt es einige **Komponenten** und **Übersetzungsschritte**



- **Höhere Programmiersprachen** sind Programmiersprachen der 3. Generation
  - Programmausdrücke nah an der englischen Sprache
  - Hardware-unabhängig
  - Hohe Produktivität und gute Lesbarkeit
- Der **Compiler** bzw. **Übersetzer**...
  - analysiert Quelltext (Syntax und Semantik) auf Korrektheit
  - übersetzt den Quelltext in eine Zielsprache
    - Meist in Assemblersprache, Bytecode (Befehle für eine virtuelle Maschine) oder direkt in Maschinensprache
- Kompilierung = Vorgang der Übersetzung

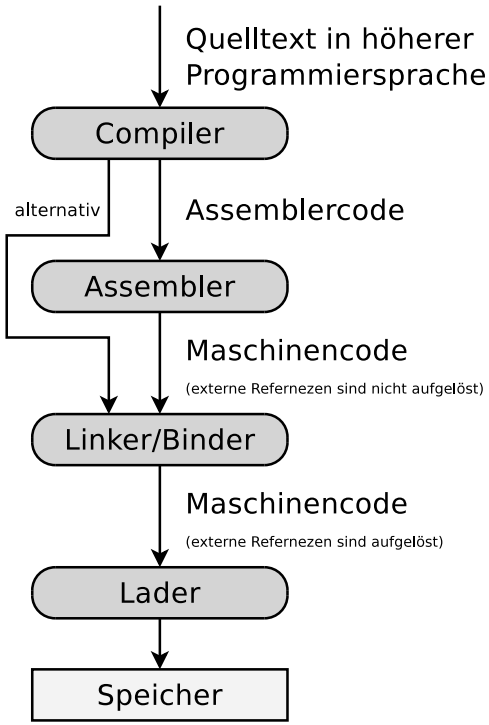
Weg vom Programm zum Maschinenprogramm (2/3)

- **Assemblersprachen** sind Programmiersprachen der 2. Generation
  - Besser lesbare Form der Maschinensprache
  - Jede Computerarchitektur hat ihre eigene Assemblersprache



- Der **Assembler** übersetzt die maschinennahe Assemblersprache in Maschinensprache
- **Maschinensprachen** sind Programmiersprachen der 1. Generation
  - Programme sind lückenlose Folgen von Nullen und Einsen
  - Für Menschen kaum interpretierbar
- Ein **Disassembler** kann Maschinensprache in Assemblersprache rückübersetzen
  - Bezeichner und Kommentare können nicht wiederhergestellt werden, da sie durch die Assemblierung verloren gehen

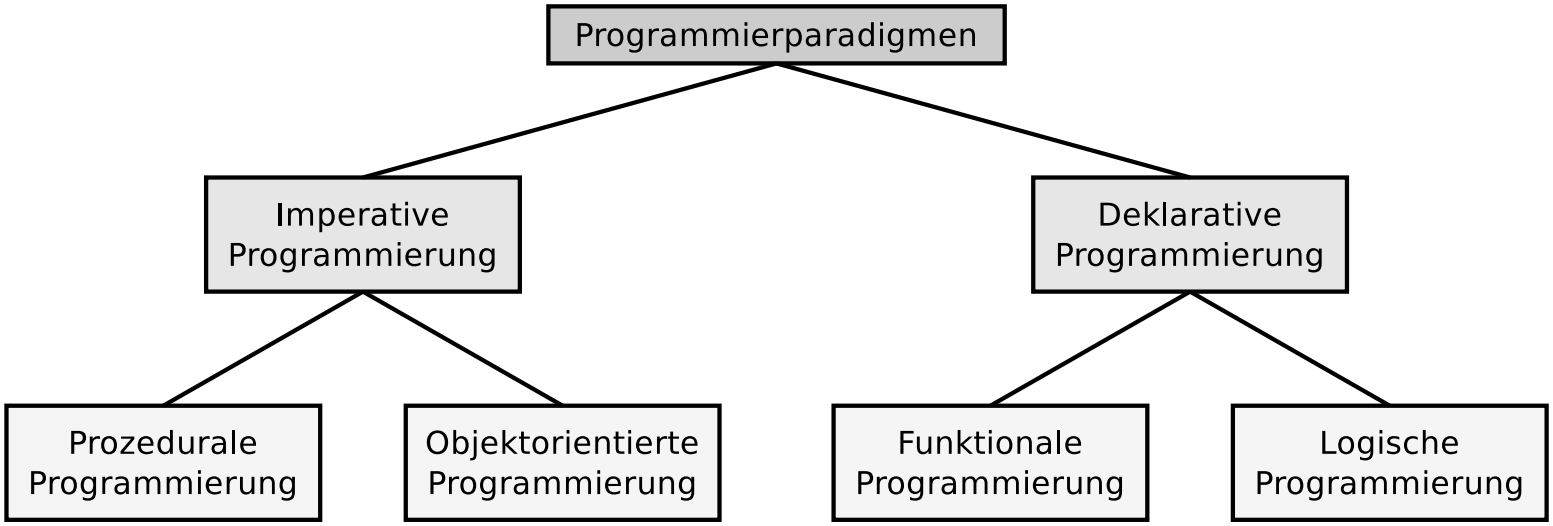
Weg vom Programm zum Maschinenprogramm (3/3)



- Der **Linker** bzw. **Binder** verbindet die einzelnen Programmmodule zu einem ausführbaren Programm
  - Programme enthalten häufig Bestandteile (Module), die auch in anderen Programmen Verwendung finden können
  - Mehrere kompilierte Module mit Funktionen (Objektdateien) können zu Funktionsbibliotheken (Programmbibliotheken) zusammengefasst werden
  - Der Linker fügt den Quelltext der Module zum Hauptprogramm hinzu, falls eine entsprechende Funktion benötigt wird
- Der **Lader** (*Loader*) lädt ausführbare Programme in den Arbeitsspeicher und führt diese aus

Programmierparadigmen

- Programmiersprachen lassen sich in verschiedene Kategorien einteilen



- Eine Programmiersprache kann mehreren Paradigmen gehorchen, also die begriffsbestimmenden Merkmale mehrerer Paradigmen unterstützen

Sehr gute deutschsprachige Quelle zu Programmierparadigmen und Programmiersprachen

Martin Grabmüller. Forschungsbericht **Multiparadigmen-Programmiersprachen**. Technische Universität Berlin. Oktober 2003.
`http://cs.tu-berlin.de/cs/ifb/Ahmed/RoteReihe/2003/TR2003_15.pdf`

Imperative Programmierung

- Bekanntestes Programmierparadigma
- Verwendet **zustandsbehaftete Programmiersprachen**
- Jedes Programm besitzt bei der Ausführung einen Zustand, der während des Programmablaufs durch Anweisungen modifiziert wird
- Programme bestehen aus Anweisungen (*imperare* = befehlen), in welcher Reihenfolge etwas vom Computer getan werden soll
- Anweisungen beschreiben, wie das Programm seine Ergebnisse erzeugt
- Steuerung der Befehlsausführung mit Kontrollstrukturen (Sequenz, Alternative, Schleife)
- Der Programmzustand besteht u.a. aus dem Arbeitsspeicher, auf dem das Programm arbeitet und den Peripherie-Geräten
- Die Modifikation des Zustandes wird durch Seiteneffekte (Verändern von Speicherzellen, Ein-/Ausgabe) bewirkt
- Beispiele: Fortran, Pascal, C,...

Prozedurale Programmierung

- Erweiterung der imperativen Programmierung
- Mit Prozeduren zerlegt man Programme in überschaubare Teile (Unterprogramme), die via einer definierten Schnittstelle aufrufbar sind
- Prozeduren nennt man auch Routinen oder Funktionen
- Fast alle imperativen Sprachen sind auch prozedural
- Beispiele: Fortran, Cobol, Algol, C, Pascal, Ada, Modula,...

Objektorientierte Programmierung

- Erweiterung der imperativen Programmierung
- Programme sind Mengen interagierender Objekte
- Objekte sind Instanzen von Klassen
- Eine Klasse ist ein abstraktes Modell (ein Bauplan) für eine Reihe ähnlicher Objekte
- Eine Klasse beschreibt Attribute (Eigenschaften) und Methoden (Verhaltensweisen) der Objekte
- Eine Klasse entspricht dem Datentyp eines Objekts
- Für die meisten heute eingesetzten imperativen Sprachen existieren objektorientierte Erweiterungen
- Beispiele: Java, Python, C++, Smalltalk, Simula, Eiffel,...

Deklarative Programmierung

- Verwendet **zustandsfreie Programmiersprachen**
- Programme enthalten Beschreibungen (*declarare* = erklären) des zu lösenden Problems
- Der Entwickler beschreibt, welche Bedingungen die Ausgabe des Programms erfüllen muss
- Lösungsweg wird automatisch ermittelt
- Deklarative Programme sind oft kürzer und leichter zu verstehen als vergleichbare imperative Programme
- Beispiele: Haskell, Erlang, LISP, Prolog, Aufbauwerkzeuge `make` und Ant, Transformationssprache XSLT, Abfragesprache SQL,...

Funktionale Programmierung

- Erweiterung der deklarativen Programmierung
- Programme bestehen ausschließlich aus Funktionen
- Funktionen sind partielle Abbildungen von Eingabedaten auf Ausgabedaten
- Eingabedaten werden nie verändert!
- Die Funktionen sind idempotent (frei von Seiteneffekten)
  - Bei jedem identischen Aufruf wird das gleiche Ergebnis zurückgeliefert
- Es werden ausschließlich Berechnungen mit Eingabedaten durchgeführt und dann neue (Zwischen-)Ergebnisse gebildet
- Funktionale Programmiersprachen bieten Sprachelemente zur Kombination und Transformation von Funktionen
- Beispiele: XSLT, `make`, Haskell, Scala,...

Logische Programmierung

- Erweiterung der deklarativen Programmierung
- Baut auf der Prädikatenlogik auf
- Besteht nicht aus einer Folge von Anweisungen, sondern aus einer Menge von Axiomen (Fakten, Annahmen und Regeln)
- Stellt der Benutzer eines Logik-Programms eine Anfrage, versucht der Interpreter die Lösungsaussage (Erfüllbarkeit oder Nichterfüllbarkeit) aus den Axiomen zu berechnen
- Während der Berechnung kann es nötig sein, mehrere alternative Berechnungsstränge abzuarbeiten, um nach einer Lösung zu suchen bzw. um alle Lösungen zu ermitteln
  - Die meisten Implementierungen realisieren dies über Tiefensuche mittels Backtracking
    - Dabei wird an jedem Wahlpunkt der Programmzustand gesichert und beim Fehlschlagen einer Berechnung wieder restauriert
- Beispiel: Prolog

Auswahl bekannter Programmiersprachen

Sprache	Klassifizierung
Pascal	imperativ, prozedural
C	imperativ, prozedural
Ada	imperativ, objektorientiert
Java	imperativ, objektorientiert
Smalltalk	imperativ, objektorientiert
Python	imperativ, objektorientiert, funktional
PHP	imperativ, prozedural, objektorientiert
C++	imperativ, prozedural, objektorientiert, funktional
Haskel	deklarativ, funktional
Prolog	deklarativ, logisch
LISP	deklarativ, prozedural, funktional
SQL	deklarativ
XSLT	deklarativ

- Python und LISP sind Multiparadigmen-Programmiersprachen
  - Diese kombinieren die Konzepte verschiedener Programmierparadigmen

PHP

- PHP ist ein rekursives Akronym für „**P**HP: **H**ypertext **P**reprocessor“
- `http://www.php.net`
- Skriptsprache zur Entwicklung dynamischer Webseiten oder Webanwendungen
- Syntax ist an C und Perl angelehnt
- Erweiterung für einen Webserver (z.B. Apache)
- Freie Software (PHP-Lizenz)
- Unterstützt zahlreiche Datenbansysteme, Internet-Protokolle und Funktionsbibliotheken

PHP-Skripte

- Ein in HTML eingebautes PHP-Skript sieht auf den ersten Blick wie ein HTML-Tag aus
- Wie bei allen HTML-Tags steht ganz am Anfang ein Kleiner-Zeichen (<) und am Ende ein Größer-Zeichen (>)
- Der Quelltext außerhalb des PHP-Tags wird ganz normal vom Webserver an den Browser weitergegeben und von diesem interpretiert
- Der Quelltext innerhalb des PHP-Tags wird vom Webserver an den PHP-Parser weitergegeben und von diesem verarbeitet
- Damit der Webserver weiß, dass es sich beim PHP-Quelltext nicht um einen fehlerhaften HTML-Tag handelt, wird nach dem Kleiner-Zeichen und vor dem Größer-Zeichen jeweils ein Fragezeichen platziert
- Nach dem öffnenden <? muss ohne Leerzeichen `php` stehen

```
<?php
    echo "Willkommen in der Welt von PHP";
?>
```

PHP installieren

- Man kann PHP und alle Komponenten von Hand installieren
- Bessere und schnellere Lösung: XAMPP
  - `http://www.apachefriends.org`
- Die XAMPP-Distribution von Kai Oswald Seidler und Kay Vogelgesang enthält u.a. Apache, PHP, MySQL, Perl, phpMyAdmin und SQLite
- Verfügbar für Windows, Linux, Mac OS X und Solaris
- Installation unter Windows ist trivial – einfach den Installer ausführen
- Unter Linux entpackt man als root die gepackte tar-Datei:
  - `tar xvfz xampp-linux-1.7.7.tar.gz -C /opt`
- Dann XAMPP starten:
  - `/opt/lampp/lampp start`
- Dann im Browser `http://localhost` oder `http://127.0.0.1` aufrufen und man sieht den XAMPP-Begrüßungsbildschirm

Erstes PHP-Skript – `listing-1.php`

- Das Listing muss im Verzeichnis `htdocs` von XAMPP legen
- Das Listing muss die Dateiendung `.php` haben
- XAMPP muss laufen
- Im Browser die Adresse `localhost` und den Pfad zur Datei eingeben
  - Öffnet man Dateien direkt mit dem Browser, also ohne *Umweg* über den Webserver (localhost), werden PHP-Dateien von diesem nicht verarbeitet

```
<HTML>
<HEAD>
    <TITLE>Hallo Welt</TITLE>
</HEAD>
<BODY>
<?php
/*
** Hier wird "Hallo Welt" ausgegeben.
*/

echo "Hallo Welt<BR>";
?>
</BODY>
</HTML>
```

Variablen und Konstanten

- Variablen
  - Eine Variable ist ein Paar, bestehend aus einem **Namen**, dem ein **Wert** zugewiesen ist
  - Der Wert kann durch weitere Zuweisungen verändert werden
  - Welcher Art der Wert (Inhalt) einer Variable ist, legt der **Datentyp** fest
    - z.B. Ziffern, Zeichen, Buchstaben, Wörter, Listen,...
  - Variablennamen beginnen immer mit einem Dollar-Zeichen ($)
  - Variablennamen beginnen mit einem Buchstaben oder Unterstrich
    - Danach folgt eine beliebige Anzahl Buchstaben, Zahlen oder Unterstriche
  - Zwischen Groß- und Kleinschreibung wird unterschieden (case-sensitive)
- Konstanten
  - Kann man nur einmal setzen
  - Erzeugt man mit dem Befehl `define`
  - Schreibt man immer in Großbuchstaben (ist eine Konvention)
  - Zwischen Groß- und Kleinschreibung wird unterschieden (case-sensitive)

Datentypen

- PHP unterstützt 6 elementare Datentypen
  - **Integer** (Ganze Zahlen)
  - **Double** (Gleitkommazahlen)
  - **String** (Zeichenketten)
  - **Array** (Listen)
  - **Object** (Objekte)
  - **Boolean** (TRUE und FALSE)

Ein Vorteil von PHP: *Implizite Typumwandlung*

- Bei PHP ist es nicht nötig, den Datentyp einer Variable anzugeben
- Das vereinfacht die Entwicklung
- PHP erkennt an der Art der Zuweisung bzw. am Inhalt einer Variable ihren Datentyp
- Will man z.B. eine Integer-Variable mit einem String verknüpfen und in dem String ist eine Zahl enthalten, dann wird PHP korrekt die beiden Zahlen miteinander verknüpfen
- Implizite Typumwandlung = automatisches Umwandeln des Datentyps einer Variablen

Integer

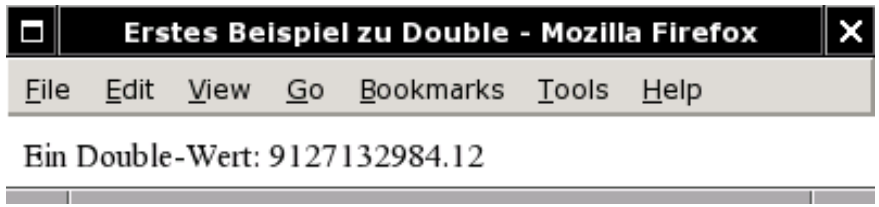
- Ein Integer ist eine ganze natürliche Zahl
- Beispiele: 9, 23, -162, 1095, 2012
- Maximale Größe eines Integers ist von der jeweiligen Rechner-Architektur abhängig
  - Gewöhnlich ist der Wertebereich ca. +/- zwei Milliarden, da ein Integer als (vorzeichenbehafteter) 32-Bit-Wert gespeichert wird

```
<?php
    $integer1 = 1231;
    echo "Ein Integer-Wert: $integer1";
?>
```

Double

- Double ist der Datentyp für Fließkommazahlen
- Fließkommazahlen sind reelle Zahlen
  - Also Zahlen mit Komma und Nachkommastellen
- Beispiele: `5.0`, `29.3.` `19674.2543`, `0.999999999`
- Achtung: Man muss das Komma als Punkt schreiben!
- Maximale Größe eines Double ist von der Rechner-Architektur abhängig
  - Da ein Double aber als 64-Bit-Wert gespeichert wird, ist eine maximale Größe von $\sim 1,8 * 10^{308}$ mit einer Genauigkeit von 14 Nachkommastellen kein Problem

```
<?php
    $Double1 = 9127132984.1231;
    echo "Ein Double-Wert: $Double1";
?>
```



String

- Ein String ist eine Zeichenkette
  - Es ist egal, um welche Zeichen es sich handelt
  - Ein String kann Buchstaben, Zahlen und Sonderzeichen enthalten
- Alle in einem String enthaltenen Zeichen werden als ein Wort angesprochen und behandelt
- Beispiele: `abcxyz`, `PHP_ist_super`, `20Euro`, `#*/[}%$`

```
<?php
    $String1 = "PHP ist toll";
    echo "$String1";
?>
```

- Es gibt *escaped characters* in Strings
  - Das sind Zeichen, die sich nach einem Backslash befinden

`\n`	*line feed* = neue Zeile	`\r`	*carriage return* = Zeilenumbruch
`\\`	Backslash-Zeichen	`\t`	Horizontaler Tabulator
`\"`	Doppeltes Anführungszeichen	`\&`	Dollar-Zeichen

Array und Object

- Array
  - Eine Reihe von Elementen des gleichen Datentyps
  - Vergleichbar einer Art Liste von Objekten
  - Auf jedes dieser Objekte kann man mittels eines Indexes zugreifen
  - Es gibt eindimensionale und mehrdimensionale Arrays
- Object
  - Bündelt Funktionen (Methoden) und Daten (Eigenschaften)
  - Kann jede Anzahl von Variablen und Funktionen enthalten
  - Ist eine Instanz einer Klasse
  - Mit Klassen lässt sich Quelltext kapseln
    - Dadurch kann man ihn störungsfrei in existierende Projekte einsetzen, ohne Gefahr, mit bereits benutzten Funktions- und Variablennamen zu kollidieren

Boolean

- Daten vom booleschen Datentyp können entweder den Wert `TRUE` oder `FALSE` haben
- Wird hauptsächlich bei Vergleichen und Schleifen angewendet

```
<?php
    if ($wert == 12): echo "es ist eine 12";
?>
```

- Inhalt einer Variable `wert` wird überprüft
- Ist dieser gleich 12, werden die Anweisungen nach dem Doppelpunkt ausgeführt
- Ansonsten wird im Code weitergegangen
- Die Abfrage wird intern von PHP mit einem booleschen Wert entschieden
- Entweder ist der Wert in der Variablen 12, dann wäre die Abfrage `TRUE`, ansonsten wäre sie `FALSE`

- Variablen können auch direkt auf `TRUE` oder `FALSE` gesetzt werden

```
<?php
    $schwarz = TRUE;

    if ($schwarz == TRUE):
        echo "Heute habe ich die schwarzen Figuren!";
    else:
        echo "Mist, schon wieder die weissen Figuren!";
    endif;
?>
```

Auswahl nützlicher Funktionen für die Arbeit mit Variablen

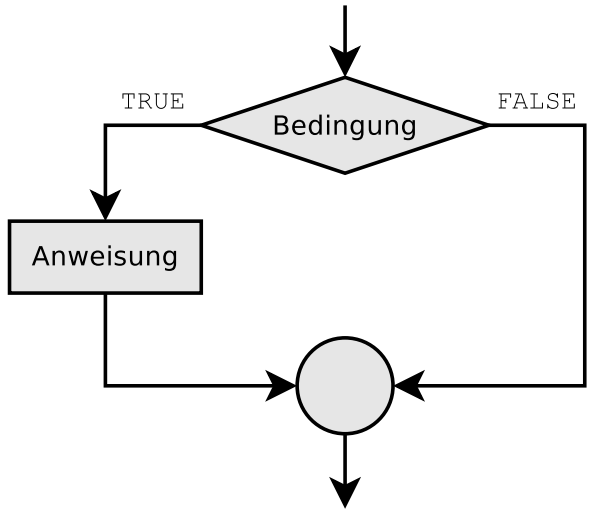
Funktion	**Funktionalität**
`unset()`	Variable oder Array aus dem Speicher entfernen
`empty()`	Prüft, ob eine Variable leer ist
`isset()`	Prüft, ob eine Variable oder ein oder Array-Element existiert
`strlen()`	Ermittelt die Länge eines Strings
`strstr()`	Sucht das erste Auftreten eines Suchstrings in einem String und liefert den Reststring zurück (beachtet Groß- und Kleinschreibung)
`stristr()`	Sucht das erste Auftreten eines Suchstrings in einem String und liefert den Reststring zurück (ignoriert Groß- und Kleinschreibung)
`strpos()`	Sucht das erste Auftreten eines Suchstrings in einem String und liefert die Position
`substr()`	Gibt einen Teil eines Strings zurück
`strrev()`	Dreht einen String um
`str_shuffle()`	Mischt den Inhalt eines Strings
`strtolower()`	Gibt einen übergebenen String in Kleinbuchstaben zurück
`strtoupper()`	Gibt einen übergebenen String in Großbuchstaben zurück
`strchr()`	Sucht das erste Auftreten eines Suchstrings in einem String und liefert den Reststring zurück (beachtet Groß- und Kleinschreibung)

Kontrollstrukturen – Grundlagen

- Steuern den Programmablauf in imperativen Programmiersprachen
- Legen die Reihenfolge der auszuführenden Aktionen fest
- Die wichtigsten Kontrollstrukturen sind:
  - **Sequenz** (Folge von Aktionen ohne jede Bedingung)
  - **Alternative/Verzweigung** (`if`, `if-else`, `elseif`)
  - **Repetition/Schleife** (`while`, `do-while`, `for`)
- Alle weiteren Kontrollstrukturen (`foreach`, `switch`, `break`, usw.) sind auf die bereits genannten zurückführbar und stellen nur eine Erweiterung dar
  - Diese Kontrollstrukturen steigern die Verständlichkeit (Lesbarkeit) des Quelltextes

if-Abfrage – einseitige Alternative

```
if (logischer Ausdruck) {
    Anweisungsblock
}
```



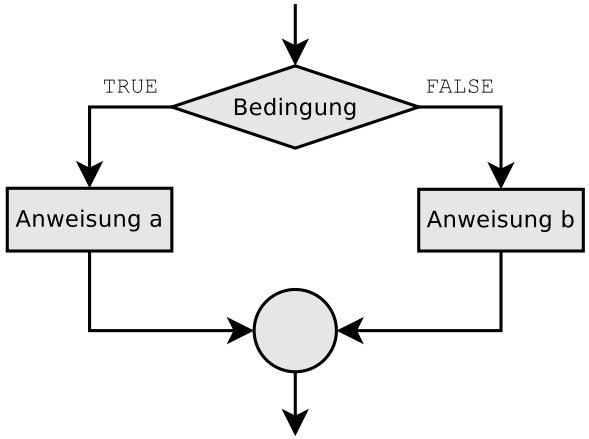
- Der logische Ausdruck wird ausgewertet
  - TRUE $\Longrightarrow$ Anweisungsblock ausführen
  - FALSE $\Longrightarrow$ Anweisungsblock überspringen
- Einfaches Beispiel

```
<?php
    $a = 4;
    $b = 3;

    if ($a > $b) {
        echo "a ist größer b";
    }
?>
```

if-else-Abfrage – zweiseitige Alternative

```
if (logischer Ausdruck) {
    Anweisungsblock
} else {
    Anweisungsblock
}
```



- Der logische Ausdruck wird ausgewertet
  - TRUE $\Longrightarrow$ Anweisungsblock a ausführen
  - FALSE $\Longrightarrow$ Anweisungsblock b ausführen
- Einfaches Beispiel

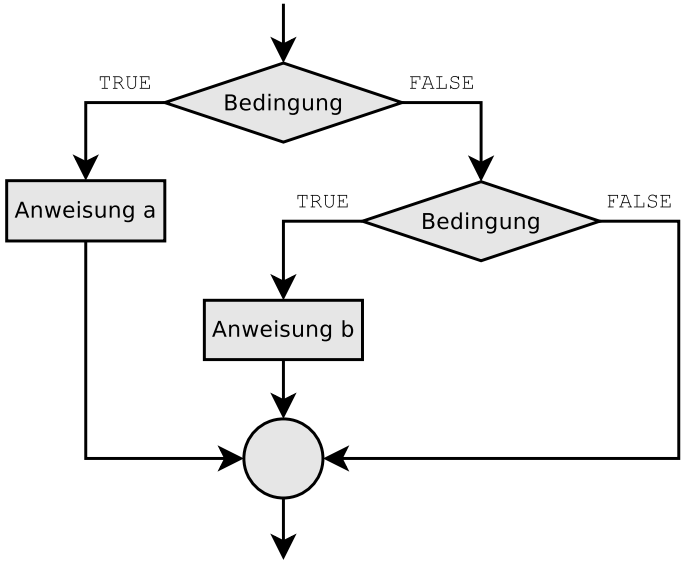
```
<?php
    $a = 4;
    $b = 3;

    if ($a > $b) {
        echo "a ist größer b";
    } else {
        echo "b ist größer a";
    }
?>
```

if-elseif-Konstrukt

- Weitere Möglichkeit der Auswertung

```
if (logischer Ausdruck) {
    Anweisungsblock
} elseif (logischer Ausdruck) {
    Anweisungsblock
}
```



- Beliebig viele elseif-Blöcke sind möglich
- So kann man noch genauer unterscheiden
- Einfaches Beispiel

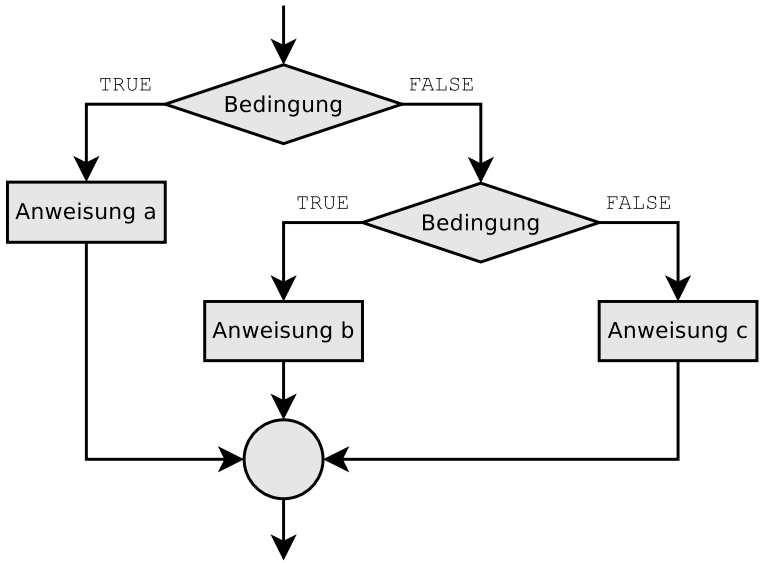
```
<?php
    $a = 4;
    $b = 3;

    if ($a > $b) {
        echo "a ist größer b";
    } elseif ($b > $a) {
        echo "b ist größer a";
    } elseif ($a == $b) {
        echo "a und b sind gleich groß";
    }
?>
```

if-elseif-else-Konstrukt

- Weitere Möglichkeit der Auswertung

```
if (logischer Ausdruck) {
    Anweisungsblock
} elseif (logischer Ausdruck) {
    Anweisungsblock
} else {
    Anweisungsblock
}
```



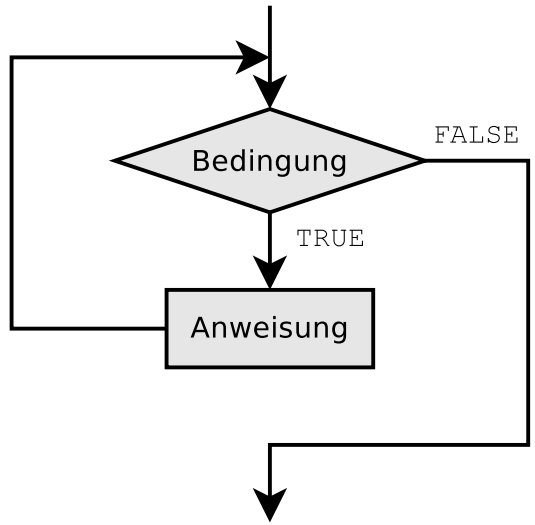
- Nutzt auch den letzten `FALSE`-Zweig
- Einfaches Beispiel

```
<?php
    $a = 4;
    $b = 3;

    if ($a > $b) {
        echo "a ist größer b";
    } elseif ($b > $a) {
        echo "b ist größer a";
    } else {
        echo "a und b sind gleich groß";
    }
?>
```

while-Schleife – abweisende Schleife

```
while (logischer Ausdruck) {
    Anweisungsblock
}
```



- Die abweisende Schleife wiederholt die Anweisung bis die Bedingung erfüllt ist
  - Eventuell wird die Anweisung nie ausgeführt, weil zuerst die Bedingung überprüft wird
- Einfaches Beispiel

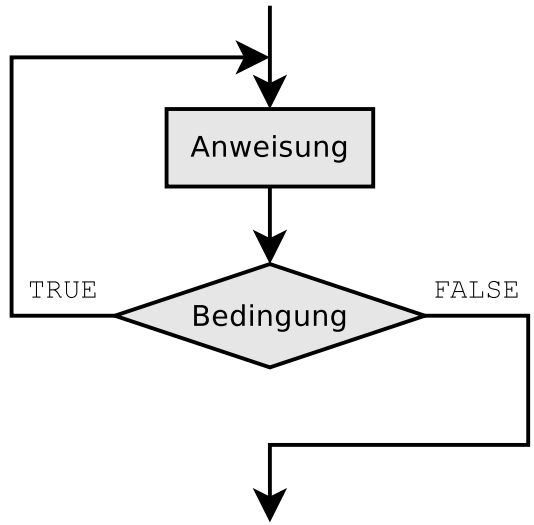
```
<?php
    $n = 0;

    echo "Ausgabe aller ganzen Zahlen, im Bereich ";
    echo "von <B>1...1000</B>, die durch <B>13</B> ";
    echo "teilbar sind.<P>";

    while ($n <= 1000) {
        if ($n % 13 == 0) {
            echo "$n ";
        }
        $n = $n + 1;
    }
?>
```

do-while-Schleife – nichtabweisende Schleife

```
do {
    Anweisungsblock
} while (logischer Ausdruck);
```



- Bei der nichtabweisenden Schleife steht die Anweisung vor der Bedingung
  - Die Aktionen im Aktionsblock werden mindestens einmal ausgeführt
- Einfaches Beispiel

```
<?php
    $n = 0;

    echo "Ausgabe aller ganzen Zahlen, im Bereich ";
    echo "von <B>1...1000</B>, die durch <B>13</B> ";
    echo "teilbar sind.<P>";

    do {
        if ($n % 13 == 0) {
            echo "$n ";
        }
        $n = $n + 1;
    } while ($n <= 1000)
?>
```

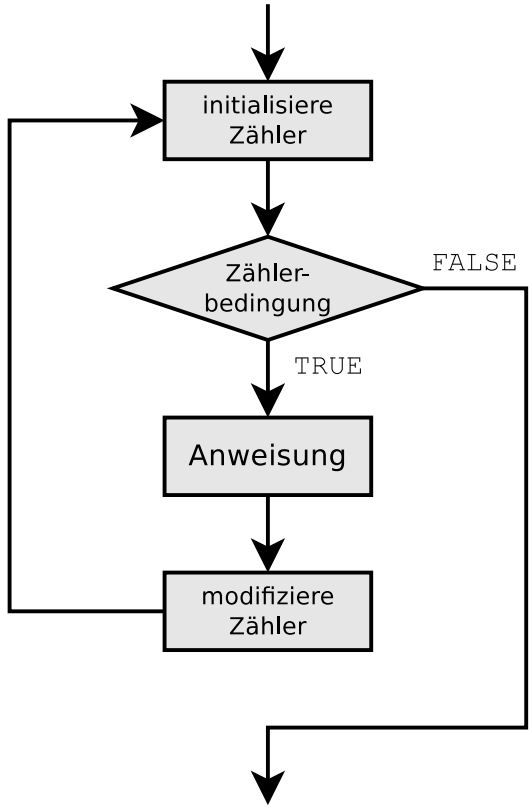
- Das Ergebnis ist identisch zu `while`

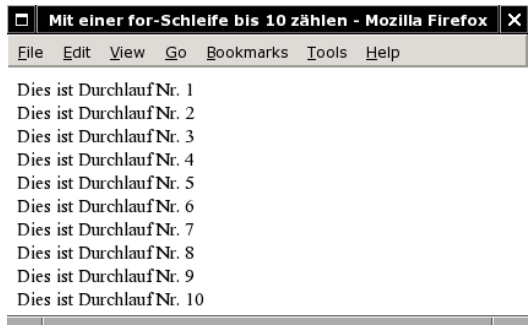
for-Schleife - Zählschleife

```
for (Zähler initialisieren; Zählerbedingung; Zähler
     modifizieren) {
    Anweisungsblock
}
```

- Führt den Anweisungsblock mit einer bestimmten Anzahl von Wiederholungen aus
- Einfaches Beispiel

```
<HTML>
<HEAD>
  <TITLE>Mit einer for-Schleife bis 10 zählen</TITLE>
</HEAD>
<BODY>
<?php
    for ($zaehler=1; $zaehler<=10; $zaehler++) {
        echo "Dies ist Durchlauf Nr. $zaehler<BR>";
    }
?>
</BODY>
</HTML>
```





Nächste Vorlesung

Nächste Vorlesung:

26.1.2012