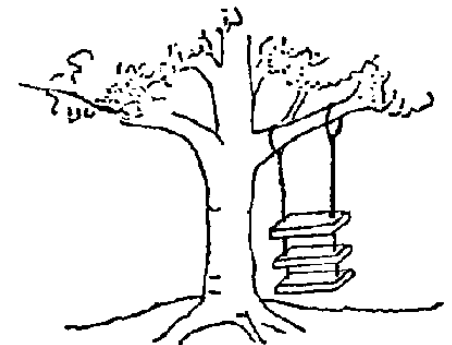


# Software Entwicklung

Ausblick SEW 1 & 2



# Inhalte des Gegenstandes SEW

## 1. Jahrgang

- Strukturierte Programmierung
  - Anweisungen & Kontrollstrukturen
  - Elementare Datentypen und Operationen
  - Prozedurale Programmierung
- Algorithmen und Datenstrukturen
  - Such und Sortieralgorithmen
- Softwareentwicklungsprozess
  - Entwicklungsumgebung
  - Testen, Fehlersuchen & Debuggen

# Ausblick SEW Jahrgang 2 bis ...

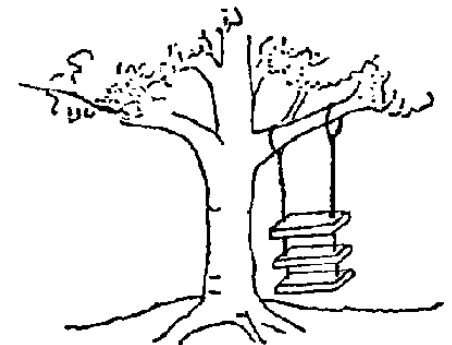
- Algorithmen und Datenstrukturen
  - Suchen & Sortieren
- Objektorientierte Programmierung
  - Objekte erstellen, UML Diagramme zeichnen
  - Programmbibliotheken nutzen
- Softwareentwicklungsprozess
  - Testfälle def., Fehler finden, Debuggen, Unit Tests
- Anwendungsentwicklung
  - Programmbausteine selbst erstellen
  - GUI für unterschiedliche Medien entwickeln

# Programmierparadigmen

- Imperative Programmierparadigmen (1. Jg)
  - Strukturierte, Prozedurale & Modulare Programmierung
  - C, Pascal, Modula-2
- Objektorientierte Programmierparadigmen (2. Jg)
  - Objekte und deren Eigenschaften wie dazugehörigen Funktionalitäten werden in Klassen zusammengefasst
  - C++, Java, C#

# Darstellung von Algorithmen

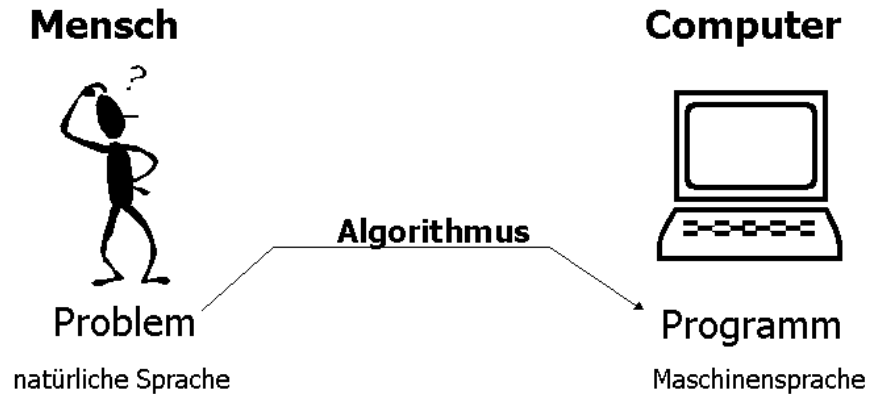
Algorithmisches Denken  
Denken in Lösungsvorschriften...



# Übersicht

- Begriffe

- Informatik
- Algorithmus



- Darstellung von Algorithmen

- Struktogramm
- Ablaufdiagramm
- Pseudocode

# Informatik

- **Donald Knuth**: Informatik (insbesondere Softwaretechnik) ist die Wissenschaft von den Algorithmen
- $\Rightarrow$  zentraler Begriff in der Softwaretechnik
- $\Rightarrow$  Algorithmen sind Abstraktionen von Programmen

# Informatik vs Mathematik:

- Mathematik: primär statisch  
( $\Rightarrow$  Aussagen über statische Zusammenhänge)
- Algorithmik: primär dynamisch  
( $\Rightarrow$  Zeit/Ablauf spielen eine Rolle)
- $\Rightarrow$  Algorithmisches Denken ist eine allgemeine Methode Wissen zu organisieren, d.h. es ist nicht auf die Softwaretechnik beschränkt



# Software Entwicklung

- **Softwaretechnik engl. Software Engineering**
  - beschäftigt sich mit der Herstellung oder Entwicklung von Software
  - der Organisation und Modellierung der zugehörigen Datenstrukturen und
  - dem Betrieb von Softwaresystemen.



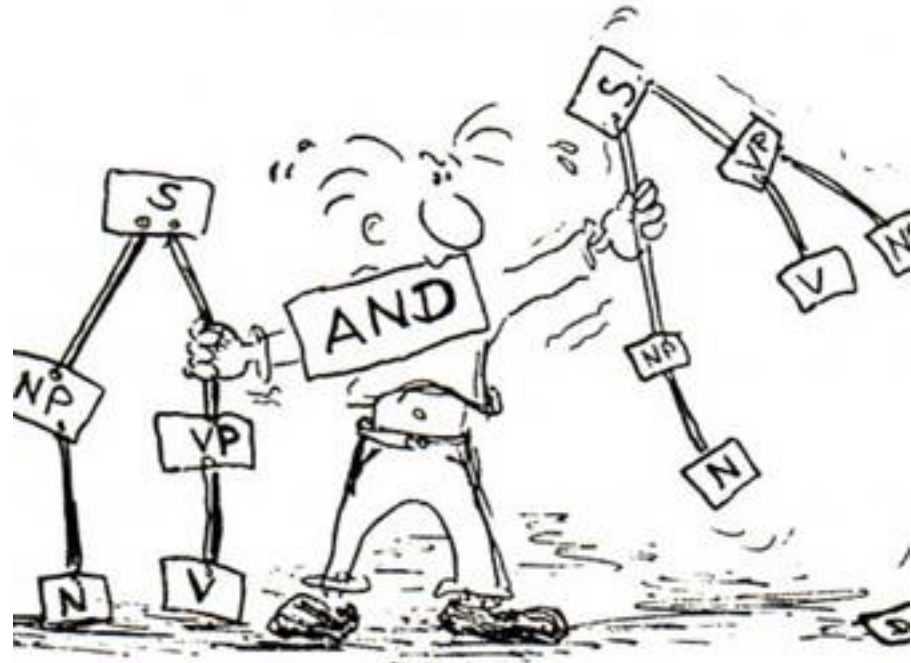
# Algorithmus

Ein Algorithmus ist eine detaillierte und explizite  
Vorschrift zur schrittweisen Lösung eines Problems.



# Programmiersprache

formale Sprache zur Formulierung von Datenstrukturen und Algorithmen, d. h. von Rechenvorschriften, die von einem Computer ausgeführt werden können, genannt Quellcode



# Syntax

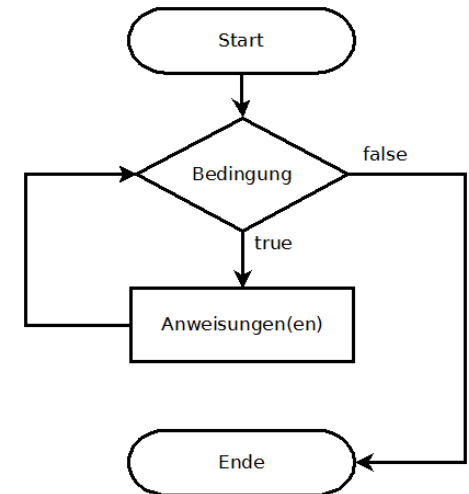
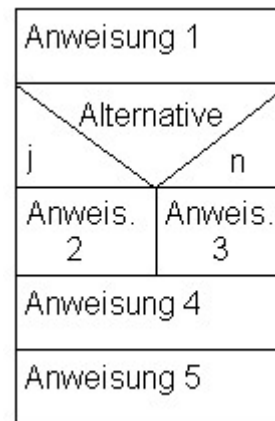
Quellcode setzt sich aus Anweisungen nach einem vorgegebenen Muster zusammen, der sogenannten **Syntax**.

# Algorithmus

- *Ein Algorithmus ist eine detaillierte und explizite Vorschrift zur schrittweisen Lösung eines Problems.*
  - Die Ausführung des Algorithmus erfolgt in einzelnen Schritten.
  - Jeder Schritt besteht aus einer einfachen und offensichtlichen Grundaktion
  - Zu jedem Zeitpunkt muss eindeutig bestimmt sein, welche Schritte als nächstes auszuführen sind.

# Algorithmen

- Algorithmen sind logische Abläufe, diese können programmiert werden oder grafisch dargestellt werden.
  - Struktogramme
  - Ablaufdiagramme
  - Programmiersprache C#
  - Pseudocode



# Darstellung von Algorithmen

als Pseudocode oder grafisch mit  
Struktogramm & Ablaufdiagramm

# Pseudocode

- Programmcode,
  - zur Veranschaulichung eines Paradigmas oder Algorithmus
  - nicht zur maschinellen Interpretation
- ähnelt höheren Programmiersprachen
  - gemischt mit natürlicher Sprache
  - und mathematischer Notation
- kann Programmablauf unabhängig von einer konkreten Programmiersprache beschreiben



# Pseudocode - Bsp Add

- Einfaches Beispiel für Pseudocode:
  - Einlesen 2er Zahlen, Ergebnis berechnen und Ausgeben in Pseudocode:

***Start Program***

***Enter two numbers, A, B***

***Add the numbers together***

***Print Sum***

***End Program***

# Pseudocode - Bsp Max Element

- Finde die Maximale Zahl in einem Array (Sammlung) von Zahlen in Pseudocode:

**Algorithm** arrayMax(A,n) :

*Input: Ein Array A, der n Integerwerte enthält*

*Output: Das maximale Element in A*

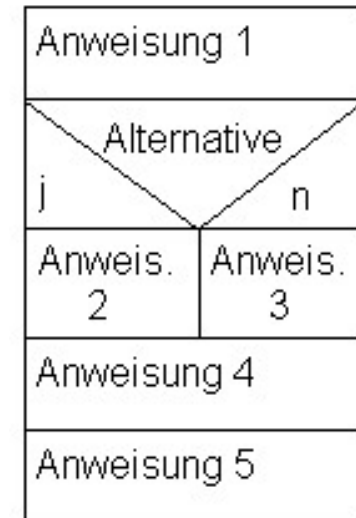
currentMax = A[0]

**for** i = 1 **to** n - 1 **do**

**if** currentMax < A[i] **then**

        currentMax = A[i]

**return** currentMax



# Struktogramm

zum Darstellen von Algorithmen, sprich die Logik eines Programms bzw den Programmfluss veranschaulichen

70er Jahren des vergangenen

Jahrhunderts von **Isaac Nassi** und **Ben Shneidermann**

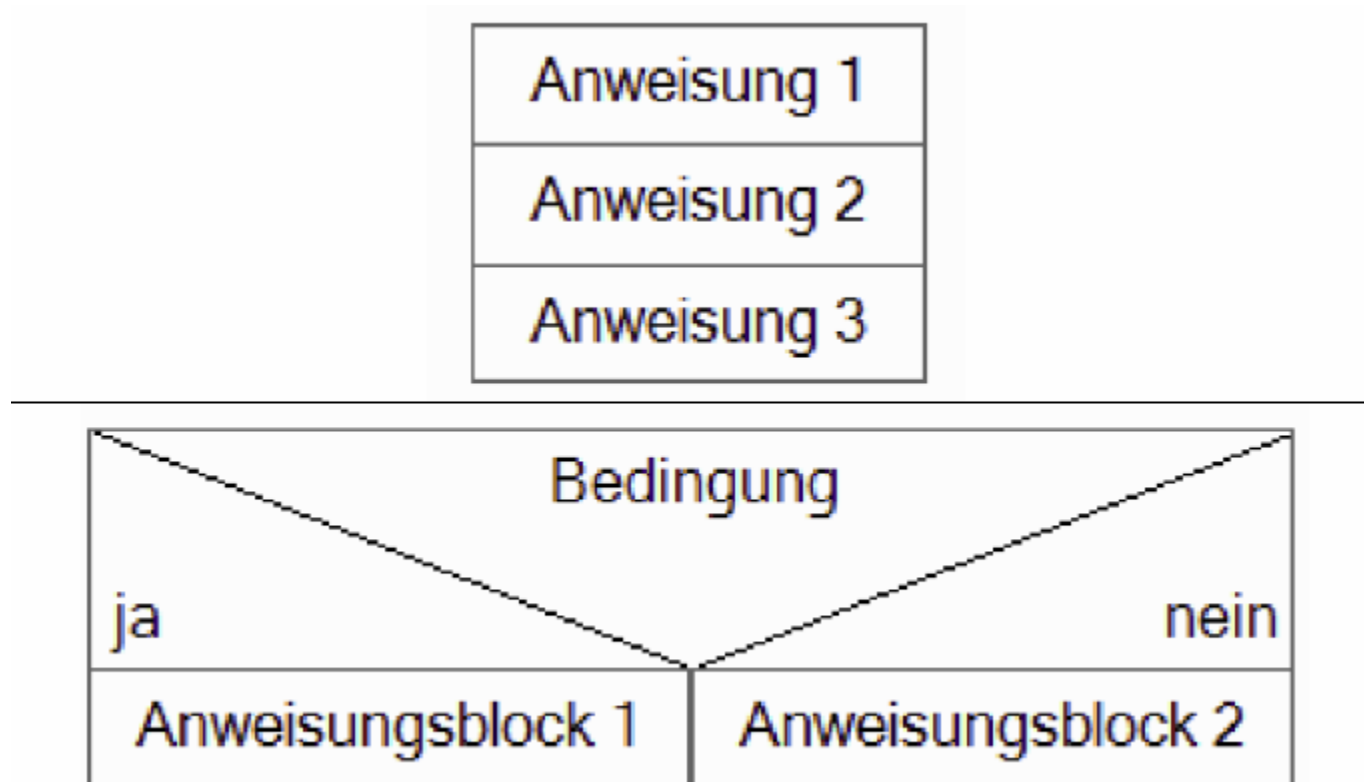
# Struktogramm

- Ablauf eines Computerprogramms auf dem Papier darzustellen
- Programmabläufe ohne Sprunganweisungen darzustellen
- bezeichnete als **strukturierte Programmierung**
  - In der professionellen Softwareentwicklung werden Struktogramme eher selten eingesetzt
  - Alternativ werden Aktivitätsdiagramme der UML (unified modelling language) verwendet

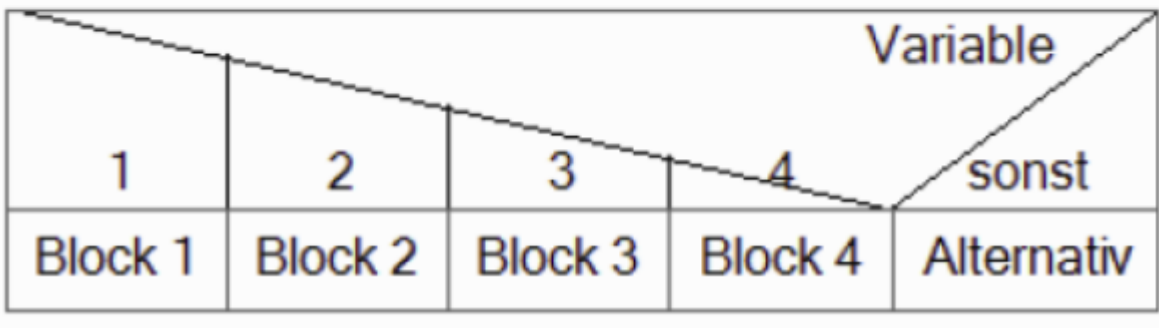
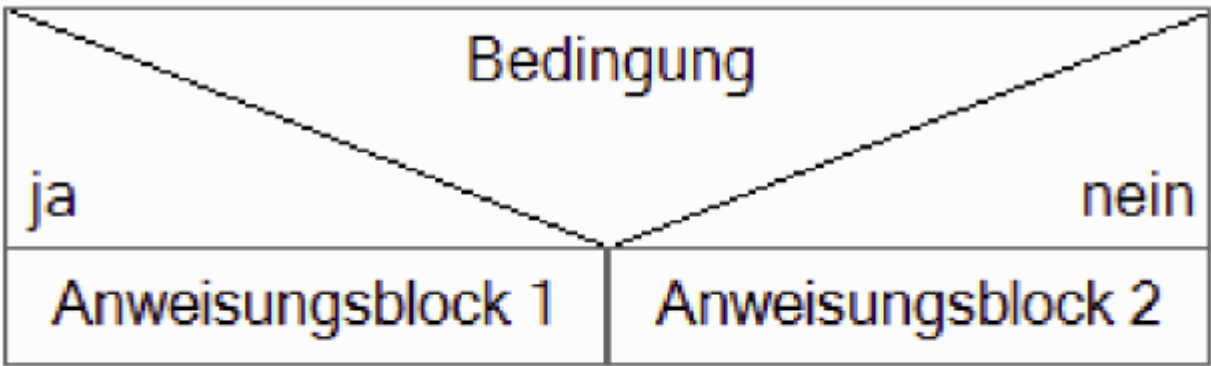
# Struktogramme

- sollten keine programmiersprachenspezifische Befehlssyntax enthalten
- müssen so programmiersprachenunabhängig formuliert werden,
  - dass die dargestellte Logik einfach zu verstehen und als Codiervorschrift in jede beliebige Programmiersprache umzusetzen ist

# Anweisungen und Verzweigungen



# If else vs Switch case



# Kopf und Fußgesteuerte Schleifen

so lange Bedingung wahr

Anweisungsblock 1

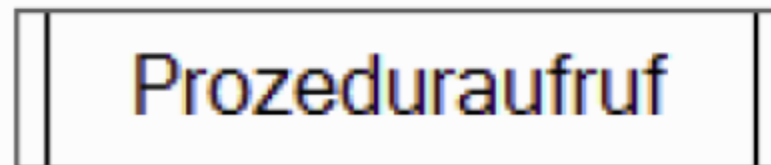
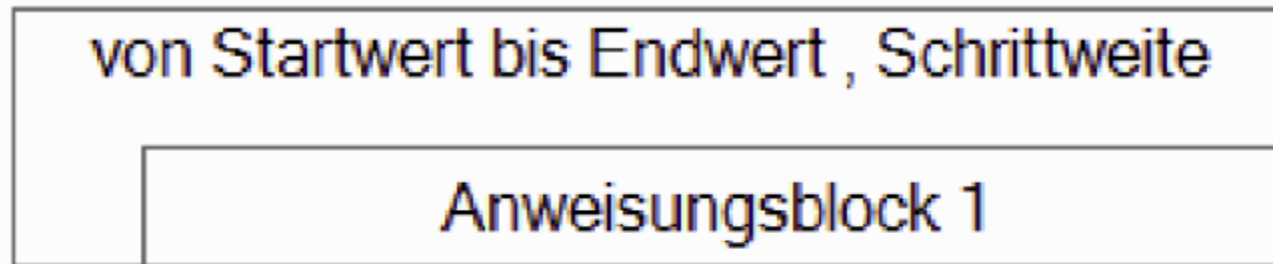
---

Anwesiungsblock 1

so lange Bedingung wahr



# Zählschleife & Prozeduraufruf



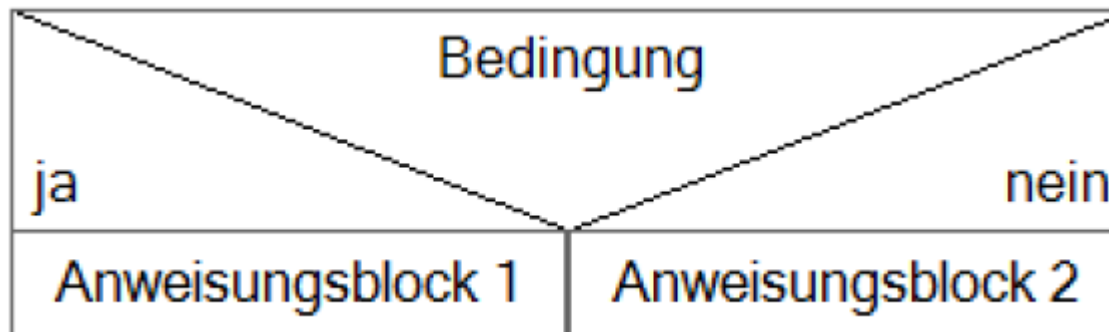
# Lineare Struktur

- Jede Anweisung wird in einem rechteckigen Strukturblock geschrieben

Anweisung 1
Anweisung 2
Anweisung 3

# Verzweigung

- Wenn eine Bedingung zutrifft wird der ja-Block ausgeführt, wenn nicht, wird der nein-Block ausgeführt.
- Die beiden Blöcke können aus mehreren Anweisungen bestehen oder können im nein-Fall auch leer bleiben.



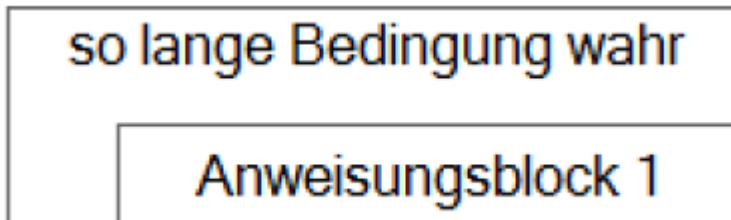
# Fallauswahl - Mehrfachauswahl

- Anhand des Zustandes einer Variablen wird einer von mehreren Anweisungsblöcken ausgeführt.
- Trifft keiner der Fälle zu, kann es einen Alternativblock geben.

Variable				
1	2	3	4	sonst
Block 1	Block 2	Block 3	Block 4	Alternativ

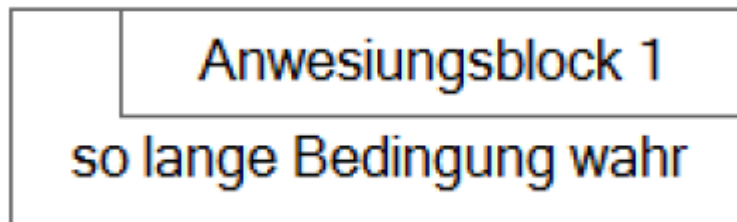
# Kopfgesteuerte Schleife

- Der Anweisungsblock wird so lange durchlaufen, wie die Bedingung zutrifft



# Fußgesteuerte Schleife

- Im Gegensatz zur kopfgesteuerten Schleife wird der Anweisungsblock hier mindestens einmal durchlaufen, weil die Bedingungsprüfung erst im Anschluss an den Anweisungsblock stattfindet.



# Zählergesteuerte Schleife

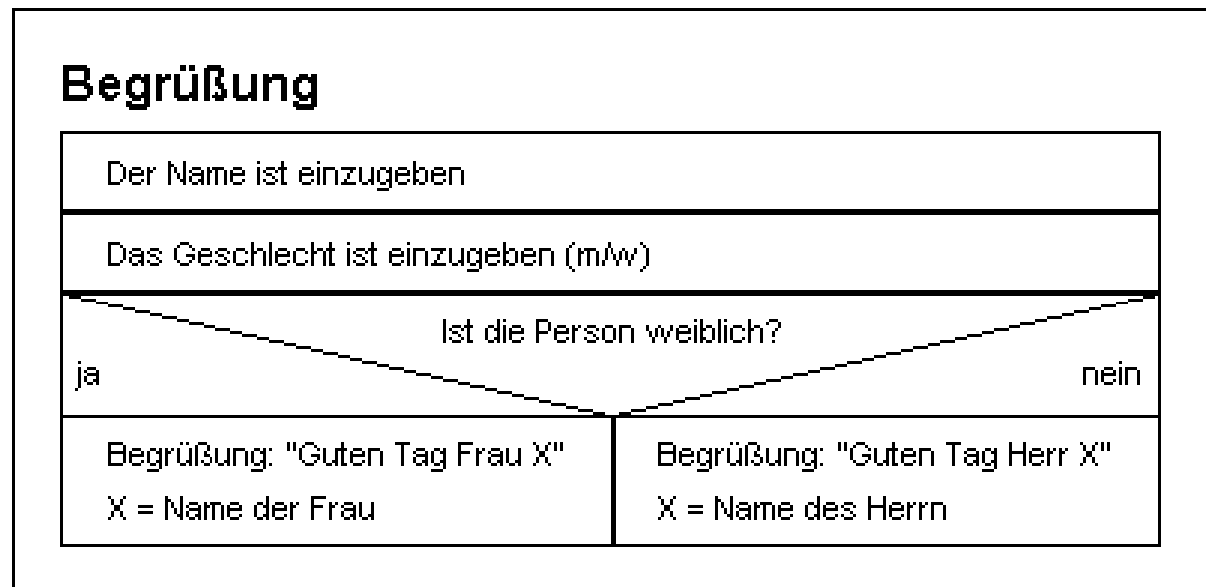
- Die Anzahl der Schleifendurchläufe wird durch eine Zählvariable festgelegt. Im Schleifenkopf werden der Startwert der Zählvariablen, der Endwert und die Veränderung der Zählvariablen nach jedem Schleifendurchlauf angegeben.

von Startwert bis Endwert , Schrittweite

Anweisungsblock 1

# Beispiel Begrüßung

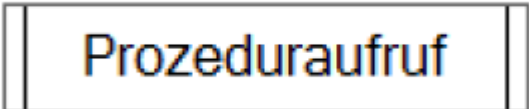
- Erstelle ein Struktogramm für
  - Eingabe des Namens
  - Auswertung ob m/w
  - Demnach „Guten Tag Frau X“ oder „Guten Tag Herr X“ ausgeben





# Prozeduraufruf

- Der Aufruf einer Prozedur oder einer Methode, die wiederum aus einer Menge von Anweisungen bestehen kann, wird durch die Doppelstriche am Rand des Strukturblocks dargestellt.



Prozeduraufruf

The diagram shows a rectangular box representing a structural block. Inside the box, the text 'Prozeduraufruf' is written. The text is preceded by a double vertical line (||) on the left side, which is the notation for a procedure call in structural modeling.

# Notenberechnung

- Erstelle ein Struktogramm, welches 3 Notenwerte einliest
- anschließend den Mittelwert berechnet
- wenn der Mittelwert, positiv ( $\leq 4$ ) ist, dann „Sie haben bestanden“ ausgeben  
sonst „Sie haben nicht bestanden“ ausgeben
- Zum Schluss soll die Ausgabe des Durchschnittswerts getätigt werden.

# Notenberechnung Lösung

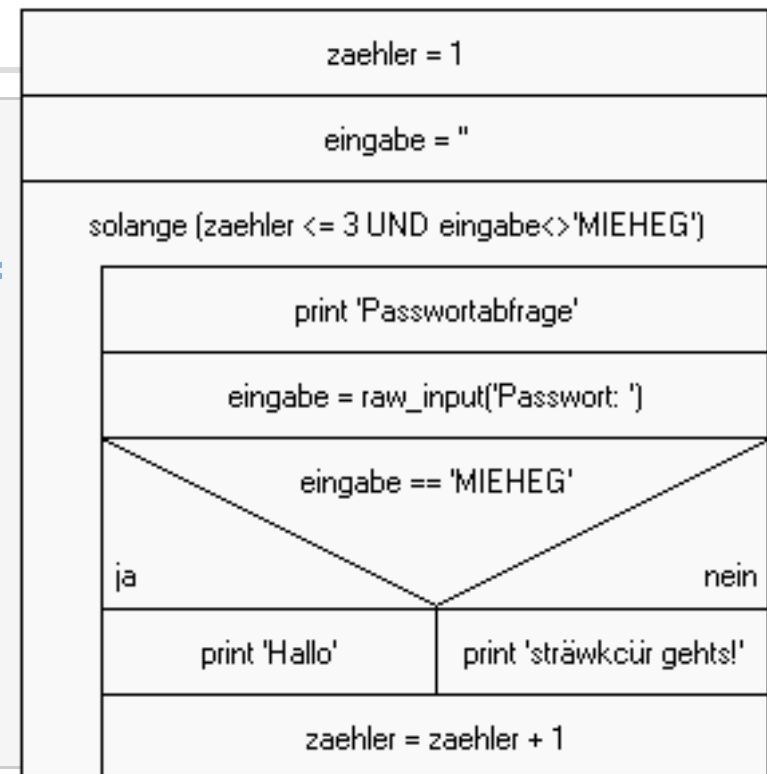
Notenberechnung	
Eingabe Note 1	
Eingabe Note 2	
Eingabe Note 3	
Verarbeitung: Mittelwert berechnen, Formel: $\text{Mittelwert} = (\text{Note1} + \text{Note2} + \text{Note3}) / 3$	
Mittelwert $\leq 4$ ?	
ja	nein
Ausgabe: Sie haben bestanden!	Ausgabe: Sie haben nicht bestanden!
Ausgabe der Durchschnittsnote (Mittelwert)	

# Passwortabfrage

- Erstelle ein Struktogramm für folgende Passwortabfrage:

## Python-Programm

```
zaehler = 1
eingabe = ''
while (zaehler <= 3 and eingabe <> 'MIEHEG'):
    # Einrückungen beachten!
    print 'Passwortabfrage'
    eingabe = raw_input('Passwort: ')
    if (eingabe == 'MIEHEG'):
        print 'Hallo'
    else:
        print 'sträwkcür gehts!'
    zaehler = zaehler + 1
```

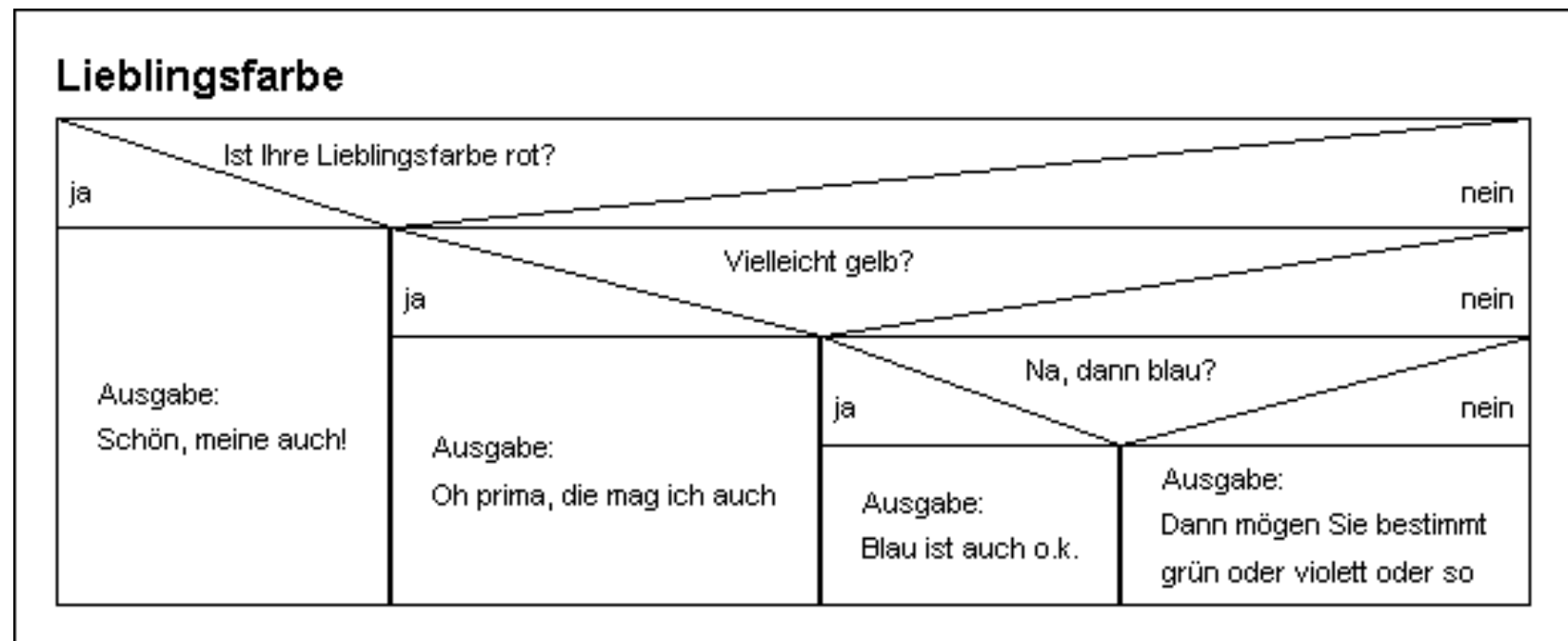


# Lieblingsfarbe

- Erstellen Sie folgendes Struktogramm
  - **Ist Ihre Lieblingsfarbe Rot?**
    - Ja - Schön meine Auch!
    - Nein - **Vielleicht gelb?**
      - Ja - Oh Prima, die mag ich auch
      - Nein - **Na dann blau?**
        - Ja - Blau ist auch o.k.
        - Nein - Dann mögen Sie bestimmt grün oder violett oder so

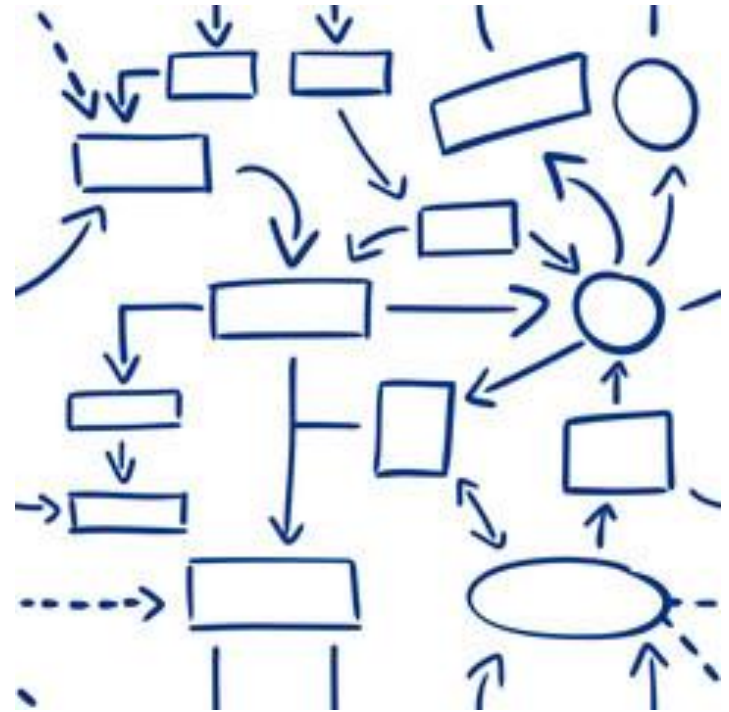
# Lieblingsfarbe

- Als Struktogramm



# Ablaufdiagramm

Flussdiagramm oder  
Programmablauf (engl flowchart)



# Ablaufdiagramm

- Programmablaufplan (PAP) ist ein Ablaufdiagramm für ein Computerprogramm,
  - auch als Flussdiagramm (engl. flowchart) oder Programmstrukturplan bezeichnet
- grafische Darstellung zur Umsetzung eines Algorithmus in einem Programm
- beschreibt die Folge von Operationen zur Lösung einer Aufgabe



# Ablaufdiagramm

- Elemente eines Flussdiagramms:
- Start / Stop
- Anweisung
- Funktion
- Entscheidung
- Linien

Process flow line



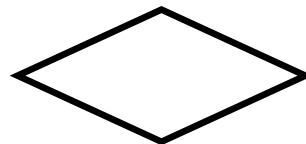
Start / End



Process step



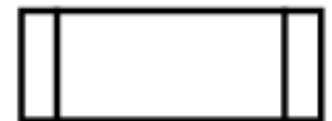
Decision



Input / Output



Predefined  
Process



# Elemente von Ablaufdiagrammen



Start oder Stopp Button



Bearbeitungsprozess, Tätigkeit



Entscheidungsfrage



Richtungsweiser, Ablauflinie



Anschlusspunkt

Software

<http://www.flowgorithm.org/download/index.htm>

# Wiederholungen



Startsymbol



Elementaraktion



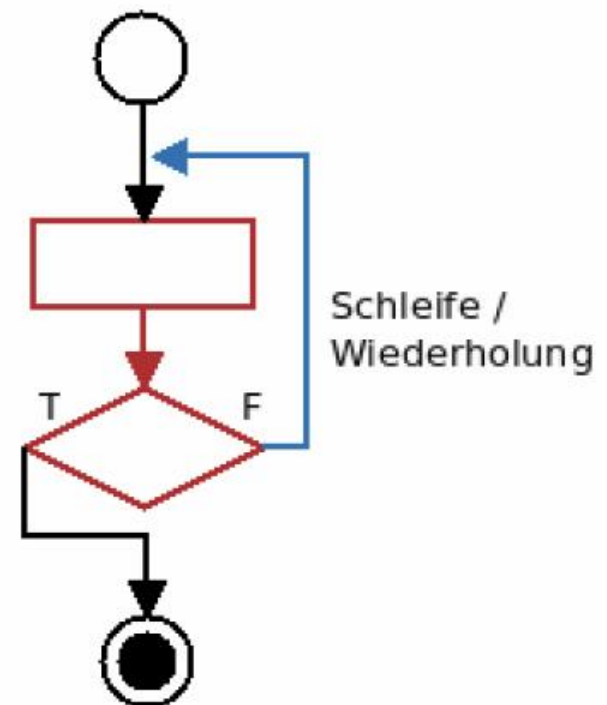
Zeigt auf nächste Aktion



Bedingung: true oder false?

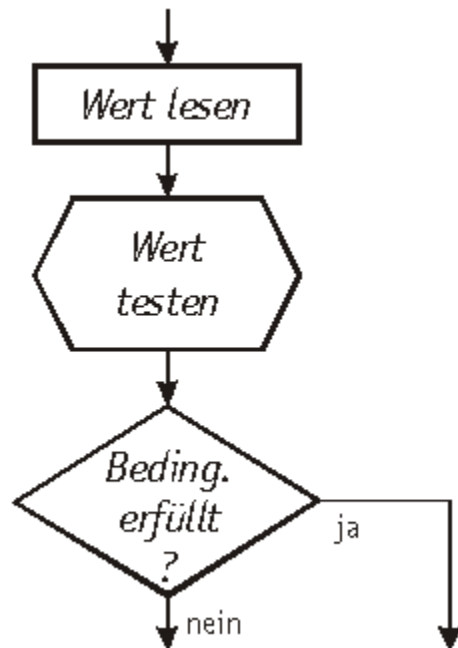


Endsymbol



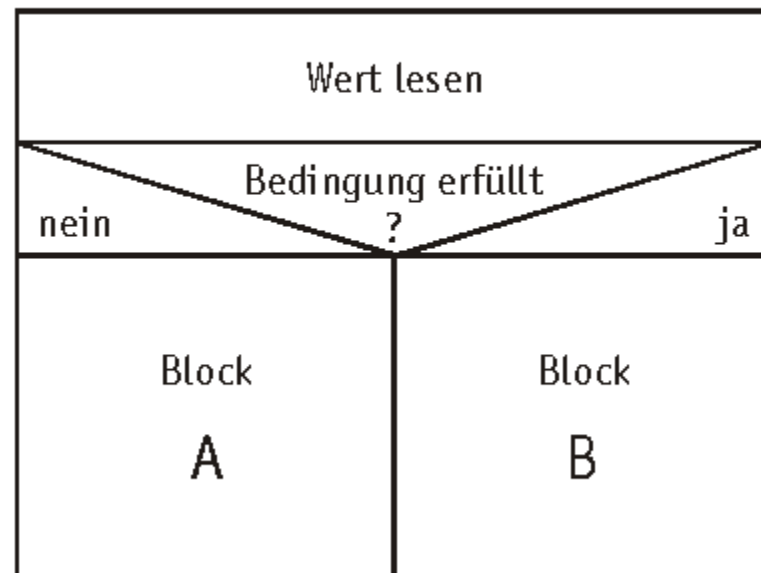
# Bedingungen

Flussdiagramm

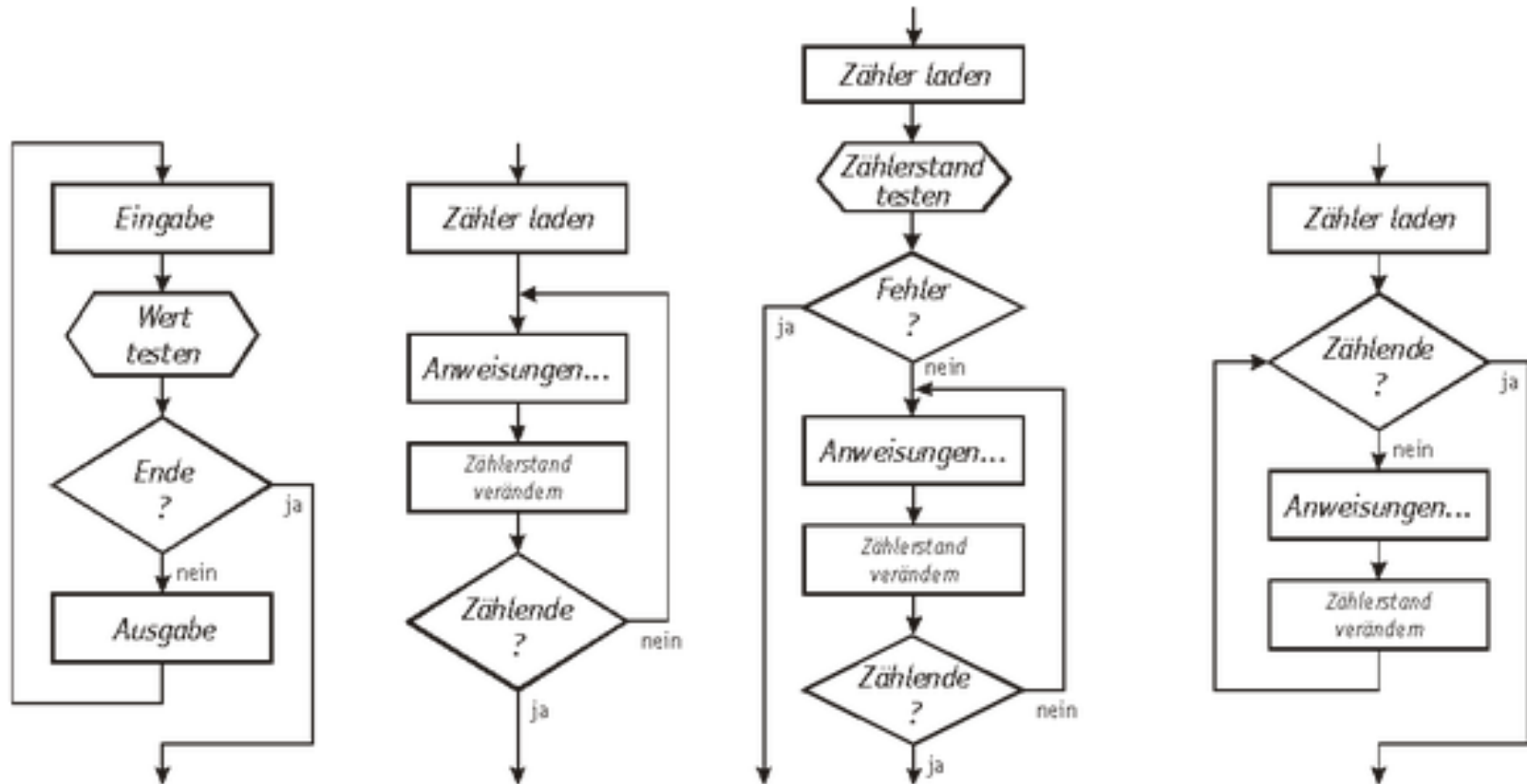


vs

Struktogramm



# Schleifen in Ablaufdiagrammen

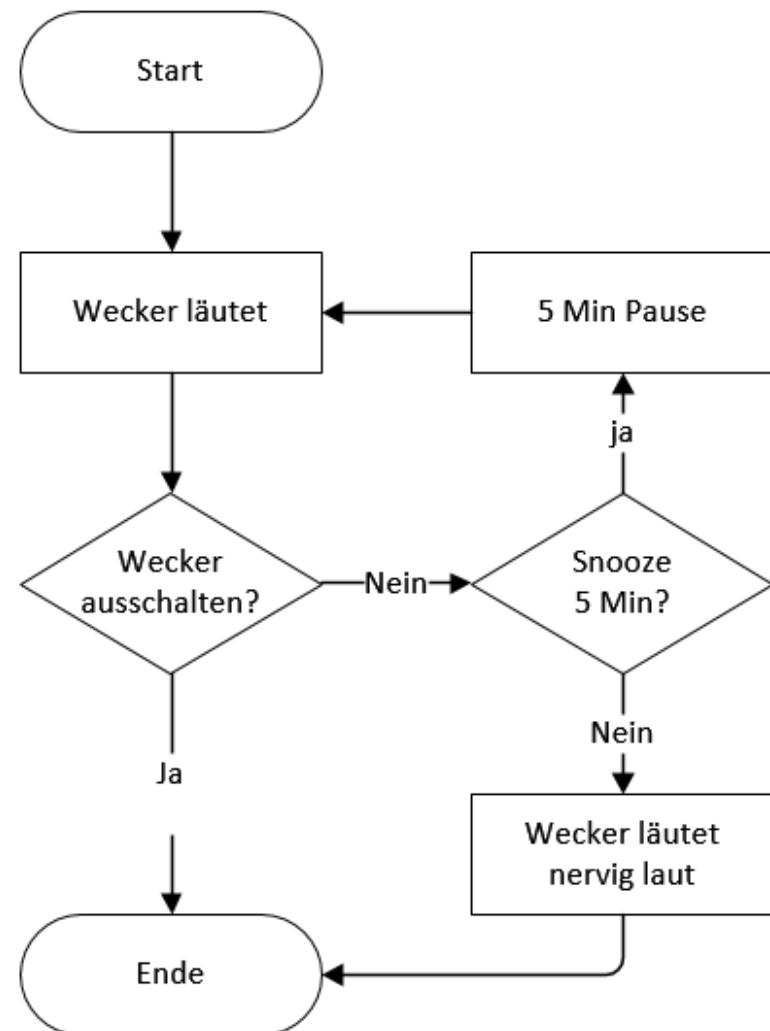


# Morgens aufstehen

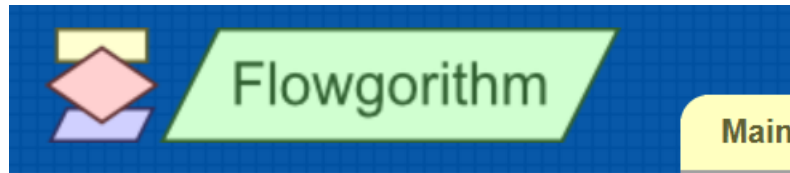
- Erstelle ein Ablaufdiagramm
  - Wecker läutet
  - Wecker ausschalten & aufstehen
  - oder
  - Snooze Button ->
    - ja: 5 Min Pause,  
bevor der Wecker wieder läutet
    - nein: Dauerläuten dann aus

# Ablaufdiagramm: Morgens aufstehen

- Wecker läutet
- Ausschalten & aufstehen
- Snooze Button: ja/nein



# Software



**Download Flowgorithm 2.22.1**

The application is available for Microsoft Windows

**Windows 10 & 8**



**Windows 64-Bit**

Uses .NET 4.6. For Windows 10 and 8.



**Windows 32-Bit (.NET 4)**

Uses .NET 4.6. For Windows 10 and 8.

- Suchen Sie ein Programm zum Erstellen von Ablaufdiagrammen und Struktogrammen, das Freeware ist und sich zum Erstellen von Diagrammen eignet.

- MS Visio

- ...

<http://www.flowgorithm.org/download/index.htm>





# Übungsbeispiele

Struktogramme und Ablaufdiagramme

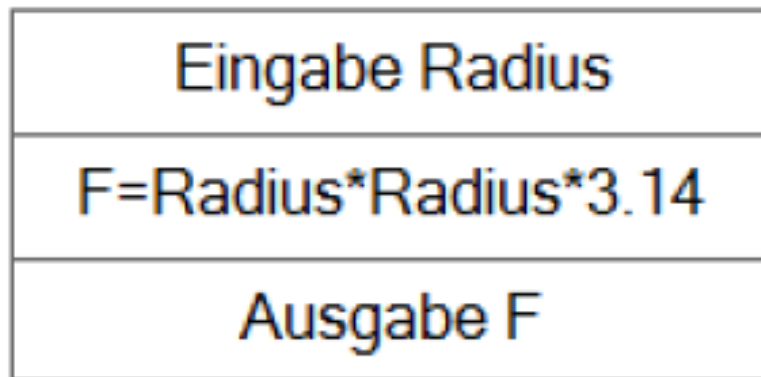
# Beispiel Kreis

- Ein Programm soll den Radius eines Kreises über die Tastatur einlesen, die Fläche berechnen und anschließend den Wert der Fläche ausgeben.

## HINWEIS:

- Bei diesem Programm handelt es sich um die klassische E-V-Ä (Eingabe-Verarbeitung-Ausgabe) - Situation.
- Alle Anweisungen werden in zeitlicher Reihenfolge ausgeführt.
- Es ist zu beachten, dass die Bezeichnungen der Variablen in allen Anweisungen korrekt verwendet werden müssen.

# Beispiel Kreis als Struktogramm



- Erstelle dieses Beispiel in C#
- Einlesen in C# mit: `Console.ReadLine()`
- Konvertieren in eine Zahl mit: `Int32.Parse(String)`
- Für PI -> `Math.PI` für genauen Wert verwenden

# Beispiel: Kreis in C#

```
|  
= namespace _01_StruktogrammBsp  
{|  
=     class Kreis  
{|  
=         public static void Main(string[] args)  
{|  
|             Console.Write("Bitte geben Sie einen Radius ein: ");  
|             String input = Console.ReadLine();  
|             Int32 radius = Int32.Parse(input);  
|             Double area = radius * radius * Math.PI;  
|             Console.WriteLine("Fläche von " + radius + " ist " + area);  
|         }  
|     }  
| }
```

# Beispiel: Multiplikation

- In diesem Beispiel wird die Variable A zu Anfang auf den Wert 5 gesetzt, die Variable B auf den Wert 3.
- Der Wert der Variable C berechnet sich aus dem Produkt aus A und B.
- Der Wert von C soll ausgegeben werden.
- Hier wird die Zahl 15 ausgegeben.

# Hinweis zu Variablen

- Variable verwendet.
  - In C# und vielen anderen Programmiersprachen müssen Variablen vor der Verwendung deklariert werden und ihr Datentyp benannt werden.
- **Deklaration**
  - stellt keine logische Anweisung dar  
wird nicht in das Struktogramm übernommen
- **Initialisierung**
  - Variable mit einem Anfangswert besetzen
  - ist eine logische Anweisung,  
die im Struktogramm auftauchen muss

# Struktogramm: Multiplikation

$A = 5$	Wertzuweisung
$B = 3$	Wertzuweisung
$C = A \times B$	Berechnung
Ausgabe C	Ausgabe

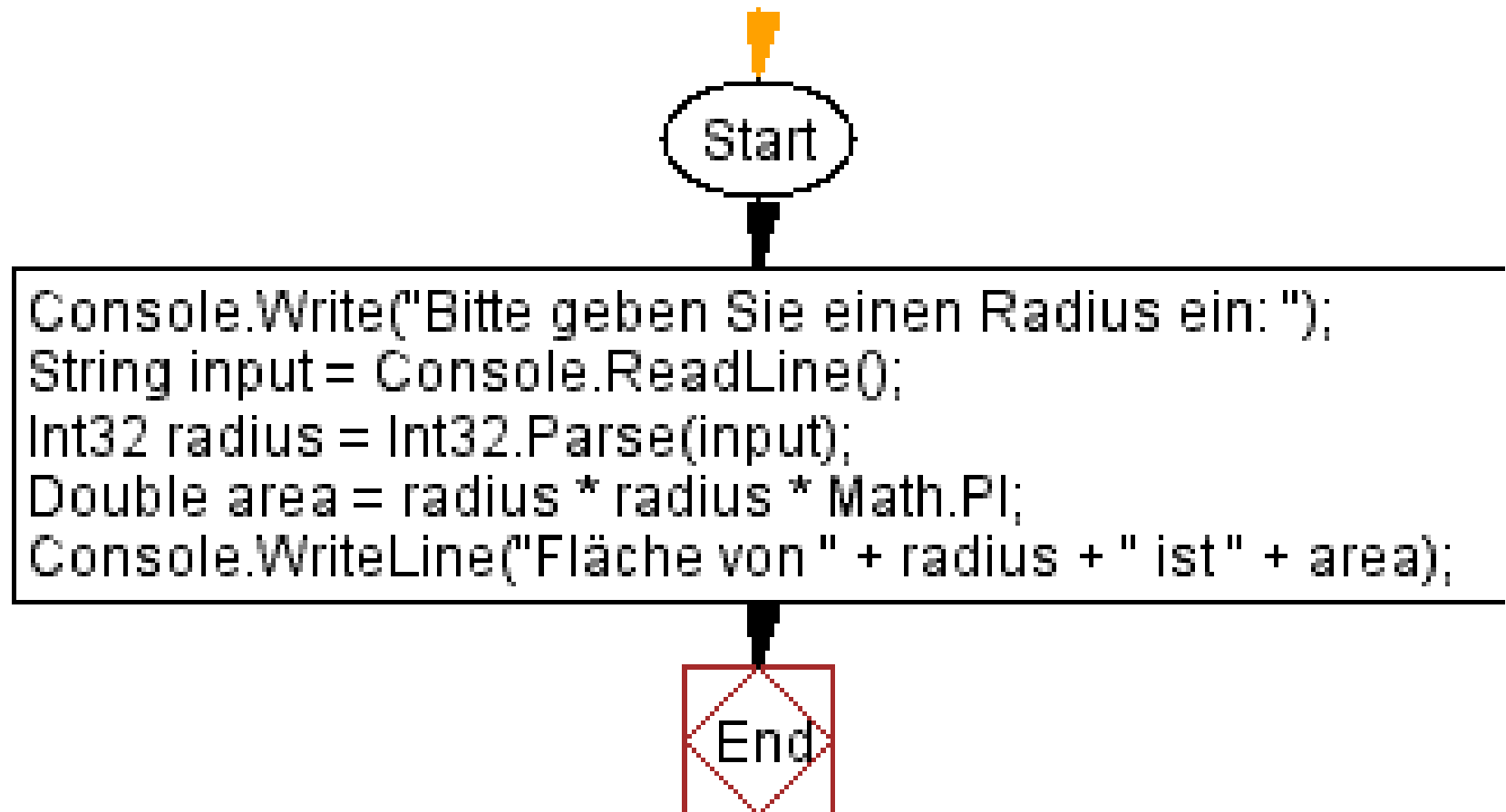
Erstelle dieses Beispiel in C# und  
Zeichne es als Flussdiagramm

# Beispiel Multiplikation in C#

```
namespace _01_Struktogramme_Bsp
{
    class Multiplication
    {
        static void Main(string[] args)
        {
            int a = 4;      int a = Int32.Parse(Console.ReadLine());
            int b = 3;
            int c = a * b;
            Console.WriteLine(a + " mal " + b + " ist " + c);
        }
    }
}
```



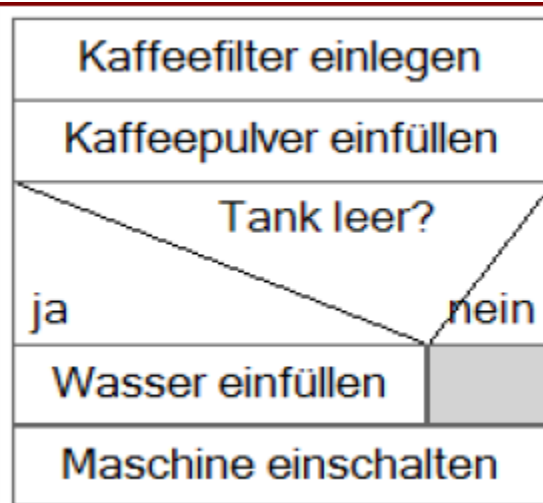
# Flussdiagramm



# Beispiel: Kaffeekochen

- Beim Kaffeekochen in einer herkömmlichen Maschine fallen folgende Tätigkeiten an:
  - Kaffeefilter einlegen
  - Kaffeepulver einfüllen
  - Prüfen, ob noch Wasser im Tank ist. Wenn
  - nein, dann Tank auffüllen
  - Maschine einschalten

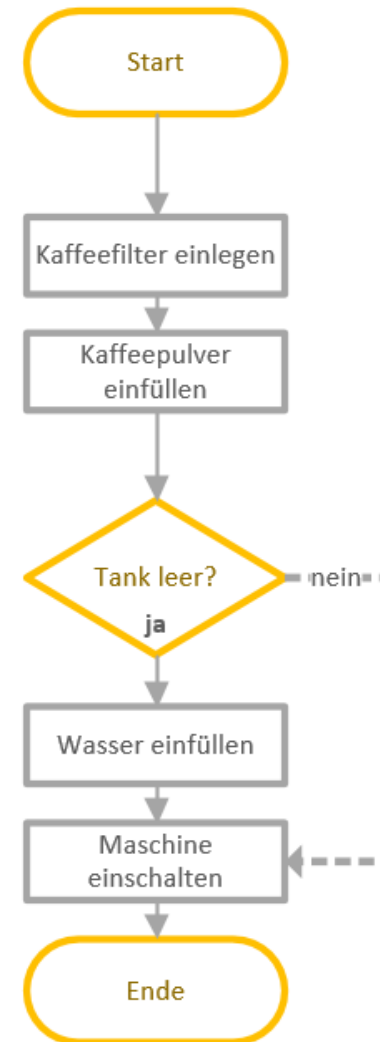
# Struktogramm von Kaffeekochen



- Bei einer Verzweigung gibt es immer zwei mögliche Fälle (Ja - Nein).
- In beiden Fällen können weitere Aktionen erfolgen.
- Ein Zweig kann aber auch (wie in diesem Beispiel) ohne Aktion bleiben.

# Kaffee-Bsp als Ablaufdiagramm

- Kaffeefilter einlegen
- Kaffeepulver einfüllen
- Prüfen, ob noch Wasser im Tank ist. Wenn
- nein, dann Tank auffüllen
- Maschine einschalten



# Kaffee Beispiel in C#

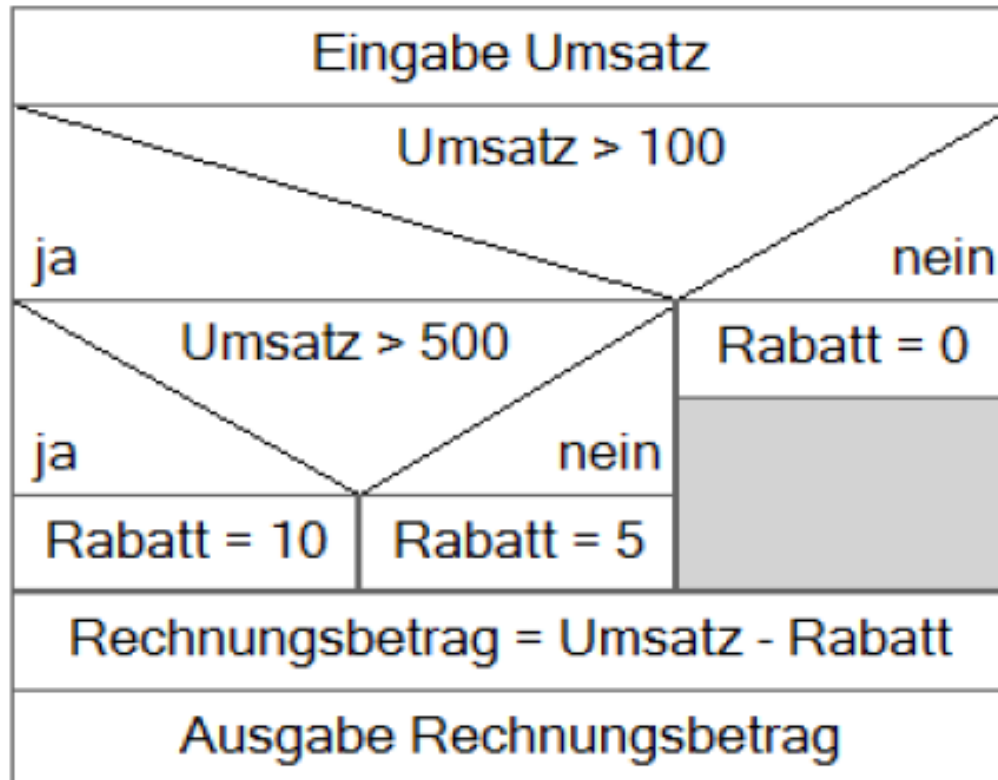
```
bool waterIsEmpty= true;

Console.WriteLine("Kaffeefilter einlegen");
Console.WriteLine("Kaffeepulver einfüllen");
if (waterIsEmpty)
{
    Console.WriteLine("Wasser einfüllen");
}
Console.WriteLine("Maschine einschalten");
```

# Beispiel: Umsatz

- Je nach Höhe des Umsatzes wird dem Kunden ein bestimmter Rabatt gewährt.
- Wenn der Umsatz höher ist als 100 €, bekommt der Kunde 5 % Rabatt.
- Beträgt der Umsatz mehr als 500 € erhält der Kunde 10 % Rabatt.
- Ein Programm soll den Rechnungsbetrag des Kunden abzüglich Rabatt berechnen.

# Struktogramm Umsatz



- In vorliegendem Beispiel wird gezeigt, dass Verzweigungen auch ineinander verschachtelt werden können.
- D. h. dass innerhalb eines ja - oder nein - Zweiges wieder eine Verzweigung folgen kann.

Erstelle ein C# Programm für dieses Beispiel.

# Umsatz in C# implementiert

```
class Sales
{
    static void Main(string[] args)
    {
        Console.WriteLine("Bitte geben Sie den Umsatz ein ");
        String input = Console.ReadLine();
        int sales = Int32.Parse(input);
        int salesDiscount = 0;
        int invoice = 0;
        if (sales > 100)
        {
            if (sales > 500)
                salesDiscount = 10;
            else
                salesDiscount = 5;
        }
        // else
        //     salesDiscount = 0;
        invoice = sales - salesDiscount;
        Console.WriteLine("Der Rechnungsbetrag ist " + invoice
            + " bei einem Umsatz von " + sales);
    }
}
```

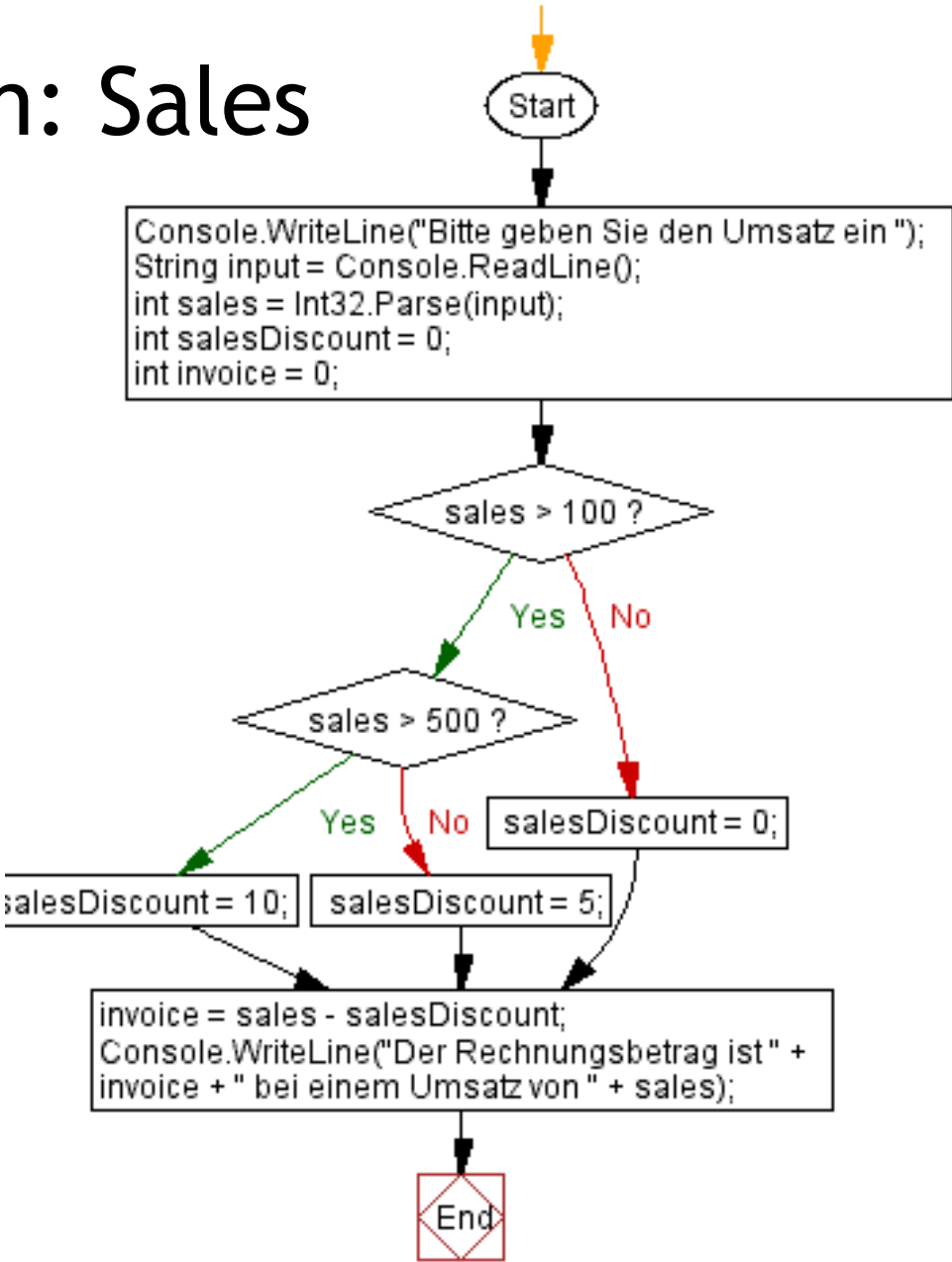
Zeichne Beispiel  
Sales als  
Ablaufdiagramm

```
Bitte geben Sie den Umsatz ein
300
Der Rechnungsbetrag ist 295 bei einem Umsatz von 300
Drücken Sie eine beliebige Taste . . .
```



# Ablaufdiagramm: Sales

Eingabe Umsatz		
Umsatz > 100		
ja	nein	
Umsatz > 500		Rabatt = 0
ja	nein	
Rabatt = 10	Rabatt = 5	
Rechnungsbetrag = Umsatz - Rabatt		
Ausgabe Rechnungsbetrag		





# Einfache Algorithmen

In C# implementieren

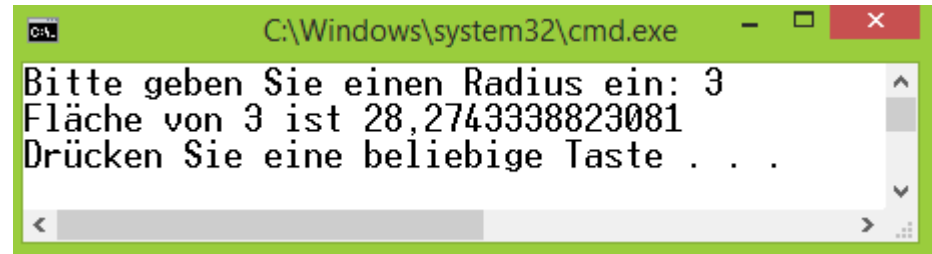
# Erste Schritte in C#

- Ein und Ausgabe in der Konsole für den Benutzer
  - `Console.Write`                      `Console.Read`
  - `Console.WriteLine`              `Console.ReadLine`
- Bedingungen
  - `If (Bedingung wahr?) then ... (else if) ... / else ...`
- Wiederholungen
  - `For / While / Do - While`
  - Solange eine Bedingung erfüllt ist, eine/mehrere Anweisungen durchführen

# Ein und Ausgabe in C#

```
namespace _01_StruktogrammBsp
{
    class Kreis
    {
        public static void Main(string[] args)
        {
            Console.Write("Bitte geben Sie einen Radius ein: ");
            String input = Console.ReadLine();
            Int32 radius = Int32.Parse(input);
            Double area = radius * radius * Math.PI;
            Console.WriteLine("Fläche von " + radius + " ist " + area);
        }
    }
}
```

# Ein- und Ausgabe in C#



```
C:\Windows\system32\cmd.exe
Bitte geben Sie einen Radius ein: 3
Fläche von 3 ist 28,2743338823081
Drücken Sie eine beliebige Taste . . .
```

- Ausgeben:
  - `Console.Write("Bitte geben Sie einen Radius ein: ");`
- Einlesen:
  - `String input = Console.ReadLine();`
- Zeichenkette in Zahl konvertieren:
  - `Int32 radius = Int32.Parse(input);`
- Berechnung
  - `Double area = radius * radius * Math.PI;`
- Ausgabe mit Zeilenumbruch
  - `Console.WriteLine("Fläche von " + radius + " ist " + area);`

# Bedingungen

- Wahrheitswert Prüfen: Bsp Kaffee

```
bool waterIsEmpty= true;  
  
Console.WriteLine("Kaffeefilter einlegen");  
Console.WriteLine("Kaffeepulver einfüllen");  
if (waterIsEmpty==true)  
{  
    Console.WriteLine("Wasser einfüllen");  
}  
Console.WriteLine("Maschine einschalten");
```

Kaffeefilter einlegen	<b>Bedingung falsch</b>
Kaffeepulver einfüllen	
Maschine einschalten	
Drücken Sie eine beliebige Taste . . .	

Kaffeefilter einlegen	<b>Bedingung true</b>
Kaffeepulver einfüllen	
Maschine einschalten	
Drücken Sie eine beliebige Taste . . .	

# Wiederholungen mit Schleifen

- Solange genug Kaffee vorhanden ist, hole einen Kaffee

[illegible]

# Schleifen

- Zählschleife:  
Zähle einen Counter von 0 bis x,
- While-Schleife:  
Solange eine Bedingung wahr ist,  
0 bis x mal ausgeführt
- Do-While  
Erledige einen Schritt solange die Bedingung  
wahr ist, 1 bis x mal ausgeführt



# Schleifen:

## For Schleife

```
public static void Main(String[] args)
{
    int amount = 10;
    for(int i = 0; i < amount; i++)
    {
        Console.WriteLine("Kaffee holen");
    }
}
```

## While Schleife

```
public static void Main(String[] args)
{
    int amount = 10;
    while (amount > 0) {
        Console.WriteLine("Kaffee holen");
        amount--;
    }
}
```

## DoWhile Schleife

```
public static void Main(String[] args)
{
    int amount = 10;
    do
    {
        Console.WriteLine("Kaffee holen");
        amount--;
    } while (amount > 0);
}
```

# Fibonacci

*Fibonacci relationship*

$$F_1 = 1$$

$$F_2 = 1$$

$$F_3 = 1 + 1 = 2$$

$$F_4 = 2 + 1 = 3$$

$$F_5 = 3 + 2 = 5$$

*In general :*

$$F_n = F_{n-1} + F_{n-2}$$

*or*

$$F_{n+1} = F_n + F_{n-1}$$

# Struktogramm

Programmname: Fibonacci.java

Beschreibung: Programm zur Berechnung und Ausgabe der

Fibonaccizahlen: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

Bemerkung: Jeweils die Summe der vorderen beiden Zahlen ergeben das Ergebnis der nachfolgenden Zahl.

Deklaration: Integer a, b, c, i

a=0

b=1

Ausgabe: a b

i=0

solange i < 6

Verarbeitung: c=a+b

Verarbeitung: a=b+c

Verarbeitung: b=a+c

Ausgabe: c a b (jeweils durch Leerzeichen getrennt (s.o.))

Erhöhe i um 1

# Fibonacci

```
static void Main(string[] args)
{
    int a = 1;
    int b = 1;
    int i = 2;
    int c = a + b;
    Console.WriteLine("Geben Sie eine Zahl ein");
    String input = Console.ReadLine();
    Int32 n = Int32.Parse(input);
    while (i < n)
    {
        c = a + b;
        a = b;
        b = c;
        i = i + 1;
    }
    Console.WriteLine("Die Fibonacci-Zahl von " + n + " ist " + c);
}
```