

# Programmierung(1)

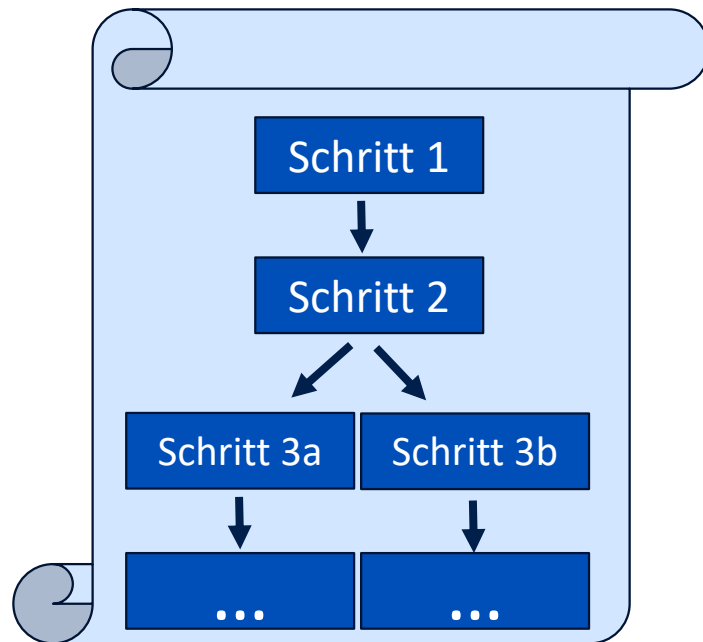
# Agenda

- Erste Grundbegriffe
  - **Programmieren** versus **Codieren**
  - **Variablen**
  - **Variable-Typ**
  - **Zuweisungs-Operator**
- Erste Fertigkeiten
  - Erstes Erstellen eines **PAP**
  - Erstes Erstellen eines **Struktogramms**
  - Erstes Erstellen eines **Pseudocodes**
- Ausführliches Training + Ergebnisbesprechung
- Fachpraktische Anwendungen

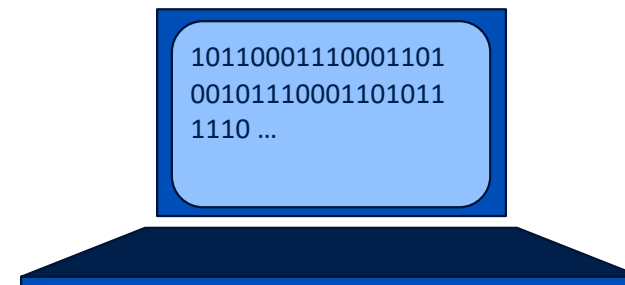
# Programmieren versus Codieren

- Werden im beruflichen Alltag oft nicht unterschieden ... **streng genommen gilt jedoch:**

„**Programmieren**“ meint das **Planen** sowie (*graphische, und/oder sprachliche*) **Beschreiben** aller Arbeitsschritte, die zur Lösung eines gegebenen Problems abzuarbeiten sind:



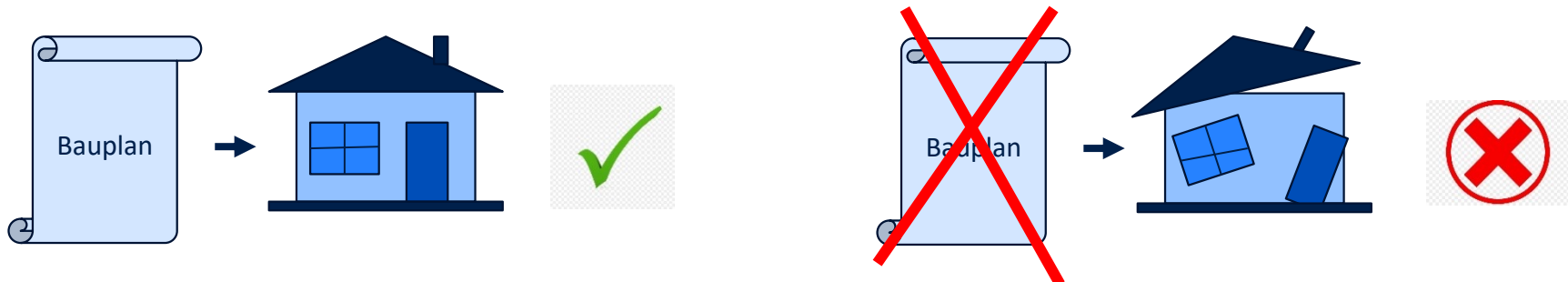
„**Codieren**“ hingegen meint das **Übersetzen** dieses graphischen Plans ... in eine für den Computer (oder Compiler / Interpreter) verständliche Sprache ... sowie das **Implementieren** (Eintragen) dieser Übersetzung in den Rechner



# Programmieren **FIRST** ... Codieren **SECOND**

- Wie im „normalen Leben“: **erst Planen – dann Ausführen**
- Mit dem Programmieren zu starten, hat folgende Vorteile:
  - Programme sind „**Sprach-übergreifend**“, also im besten Falle **international verständlich**
  - Programme sind „**Computersprachen-übergreifend**“, also in unterschiedliche (Programmier)-Sprachen übersetzbar
- Allerdings: Diese Reihenfolge wird in der **beruflichen Praxis** nicht immer konsequent durchgehalten ...  
... wir werden dies in **unserem Kurs** jedoch anders halten, denn:
  - Mit der Programmierung zu starten hilft, sich auf die Programmier-Logik zu konzentrieren
  - Ein (bereits bestehendes) Programm hilft, sich beim Codieren auf die dort geltenden Regeln zu konzentrieren
  - Das Training des (reinen) Programmierens ist für die IHK-Abschlussprüfung von großer Bedeutung, da dort (im Rahmen der ja nur schriftlich abgehaltenen Prüfung) kein Codieren möglich sein wird

- Oder kurz:



# EDV = Elektronische **Daten**-Verarbeitung **VARIABLEN**

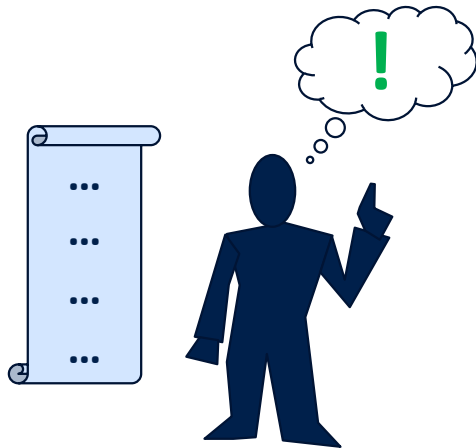
- Wer ein Programm schreibt, der wird im wesentlichen mit **Daten** arbeiten
- Um mit Daten arbeiten zu können, müssen diese „vorliegen“, also **abgespeichert** werden
- Eine **Abspeicherung** von Daten geschieht ...
  - ... **technisch betrachtet** in den „**Speicherstellen**“ des Computers
  - ... aus **Sicht des Programmierers** in sogenannten „**Variablen**“
- Um abgespeicherte Daten **auswählen**, bzw. „ansprechen“ zu können, werden ...
  - ... **technisch gesehen** alle Speicherstellen mit eindeutigen Zahlenwerten („**Adressen**“) versehen
  - ... **vom Programmierer** Variablen eingeführt, denen er selbstgewählte (**Variable**)-**Namen** gibt
- Die Bezeichnung **Variable** (=„Veränderliche“) spielt darauf an, dass sich der abgespeicherte Wert einer Variablen während des Programmablaufes verändern kann (was dann aber natürlich nicht zufällig geschieht, sondern vom Programmierer so vorgesehen ist)

# Variable - TYPEN

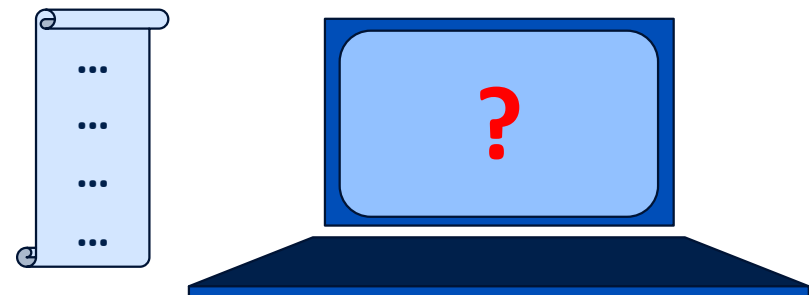
- Wenn eine Variable eingeführt wird, so muss festgelegt werden, „was“ sie speichern soll
- Dabei werden wir in diesem Baustein die folgenden 4 Variable-**Typen** unterscheiden:
  - Variable vom Typ **Integer**: ..... Variable, die (nur!) eine einzige ganze Zahl abspeichern kann
  - Variable vom Typ **Float** (oder **Double**): ..... Variable, die (nur!) eine einzige Kommazahl abspeichern kann
  - Variable vom Typ **Character**: ..... Variable, die (nur!) ein einziges Zeichen abspeichern kann  
(Beispiele sind: Buchstaben, Ziffern, Satzzeichen ...)

## Hinweise:

- a) Der menschliche Leser eines Programms kann mitdenken  
=> Variable-Typ muss **beim Programmieren** nicht ausdrücklich erwähnt werden



- b) Ein Computer kann nicht mitdenken  
=> Variable-Typ muss **beim Codieren** ausdrücklich erwähnt werden  
(damit der Rechner den jeweils notwendigen Speicherplatz reservieren kann)



# Variable – ZUWEISUNG (von „konkreten Werten“ = „Literale“)

- Wenn wir eine Variable in unser Programm einführen, so ist diese zunächst noch **undefiniert**.
- Undefinierte Variablen sind nicht „leer“ sondern besitzen einen von uns **unbeeinflussten** Wert.
- Um dies zu ändern, müssen wir den Variablen einen Startwert zuweisen. Dies geschieht durch den sogenannten **Zuweisungs-Operator**, der in den meisten Programmier-Sprachen mit dem Gleichheitszeichen-Symbol (=) dargestellt wird.

## Beispiele:

- Die Schreibweise **a=5** bedeutet, dass der Variable **a** nun der (ganzzahlige) Wert **5** zugewiesen wurde
- Die Schreibweise **preis=9.95** bedeutet, dass der Variable **preis** nun der (Komma)-Wert **9.95** zugewiesen wurde  
Hinweis: Kommazahlen müssen in der Regel (zumindest innerhalb der **Codierung**) entsprechend der angelsächsischen Schreibweise mit einem Punkt (statt Komma) notiert werden. Innerhalb der **Programmierung** ist dies nicht zwingend, aber empfehlenswert!
- Die Schreibweise **meinLieblingsZeichen='q'** bedeutet, dass der Variable **meinLieblingsZeichen** nun der (Character)-Wert **'q'** zugewiesen wurde  
Hinweis: Einzelne Zeichen müssen in der Regel (zumindest innerhalb der **Codierung**) mit „einfachen Anführungszeichen“ (= „Hochkommata“) notiert werden. Auch hier gilt: Innerhalb der **Programmierung** ist dies nicht zwingend, aber empfehlenswert!

# Variable – ZUWEISUNG (von Werten anderer Variablen)

- Wir können einer Variablen auch den Wert einer **anderen Variable** (vom selben Typ) zuweisen
- Dies macht freilich nur Sinn, wenn die zugewiesene Variable bereits einen **definierten** Wert besitzt

## Beispiele:

- Angenommen der Variable **a** wurde der Wert **22** zugewiesen:

**a=22**

dann bedeutet die Schreibweise ...

**b=a**

... dass der Variablen **b** der Wert von **a** (in diesem Fall also 22) zugewiesen wurde

=> a und b speichern nun also **den selben** Wert.

- Angenommen der Variable **zeichen1** wurde der Wert **'x'** zugewiesen:

**zeichen1='x'**

dann bedeutet die Schreibweise ...

**zeichen2=zeichen1**

... dass der Variablen **zeichen2** der Wert von **zeichen1** (in diesem Fall also 'x') zugewiesen wurde

=> zeichen1 und zeichen2 speichern nun also **den selben** Wert.



# Variable – ZUWEISUNG (von Werten, die sich durch **mathematische Operationen** ergeben)

- Wir können einer (Zahlen)-Variablen (vom Typ **Integer**, **Float** oder **Double**) auch ein **Rechenergebnis** zuweisen
- Wir wollen zu Beginn des Bausteins zunächst nur die folgenden (arithmetischen) **Operatoren** verwenden:

Operation	Symbol	Wirkung (bei Float/Double)	Wirkung (bei Integer)
<b>Addition</b> („Plus-Rechnen“)	+	wie in der Mathematik	wie in der Mathematik
<b>Subtraktion</b> („Minus-Rechnen“)	-	wie in der Mathematik	wie in der Mathematik
<b>Multiplikation</b> („Malnehmen“)	*	wie in der Mathematik	wie in der Mathematik
<b>Division</b> („Teilen“)	/	wie in der Mathematik	<b>Nur ganzzahlige Lösung OHNE REST!</b>

## Beispiele:

**a=5**  
**b=a+1** ( => b hat nun also den Wert 6)  
**c=a-3** ( => c hat nun also den Wert 2)  
**d=2\*a** ( => d hat nun also den Wert 10)  
**e=a/2** ( => e hat nun also den Wert 2)

## weitere Beispiele:

**a=3**  
**b=a+2** ( => b hat nun also den Wert 5)  
**c=a-b** ( => c hat nun also den Wert -2)  
**d=a\*a** ( => d hat nun also den Wert 9)  
**e=b/d** ( => e hat nun also den Wert 0)

## Wichtige Hinweise:

- Für jeder Art von Rechnung muss **beim Codieren** berücksichtigt werden, dass Zahlen nur bezüglich einer bestimmten **maximalen Größe** dargestellt werden können!
- Bei Rechnungen mit Kommazahlen müssen **Rundungsfehler** berücksichtigt werden!
- Beide Punkte können **beim Programmieren** ignoriert werden (und werden uns erst später beschäftigen)

# Variable – ZUWEISUNG (von „reflexiven“ Ausdrücken)

- Wir können einer (Zahlen)-Variablen auch ein **Rechenergebnis** zuweisen, in dem diese Variable **selbst** vorkommt!
- Spätestens diese reflexive Verwendung macht dann deutlich, dass der Zuweisungs-Operator **NICHT** mit einem mathematischen Gleichheitszeichen verwechselt werden darf!

## Beispiele:

**a=1**

**a=a+1** (=> a hat nun also den Wert 2)

**a=a-10** (=> a hat nun also den Wert -8)

**a=(-2)\*a** (=> a hat nun also den Wert 16)

**a=a/a** (=> a hat nun also den Wert 1)

### Erläuterung:

**a=1** (=> a hat **aktuell** den Wert 1)

**a=a+1** (=> a hat **nun** also den Wert 2)



# Variable – ZUWEISUNG (Zusammenfassung)

- Mit Hilfe des Zuweisungs-Operators kann einer Variable (erstmalig oder erneut) ein Wert zugewiesen werden
- Es können konkrete Werte (also „Literele“) zugewiesen werden
- Es können aber auch Werte anderer Variablen, bzw. die Ergebnisse aus Rechnungen zugewiesen werden
- Insbesondere kann daher einer Variable auch ein (mathematischer) Ausdruck („reflexiv“) zugewiesen werden, innerhalb dessen diese Variable selbst vorkommt:
- $a=a+1$  bedeutet dann **aber eben NICHT**, dass die linke Seite ( $a$ ) „genauso groß“ wie die rechte Seite ( $a+1$ ) sei (denn dies wäre ja auch Unsinn!), sondern meint lediglich, dass die Variable, die links steht ( $a$ ), mit jenem Wert gefüllt wird, der rechts vom Zuweisungs-Operator notiert wurde, hier also mit dem Ergebnis von  $a+1$ .

**Variable** (die gefüllt werden soll) = **Wert** (mit dem sie gefüllt werden soll)

# EVA-Prinzip: Eingabe -> Verarbeitung -> Ausgabe


- Viele Programmier-Anfänger teilen mit, dass es ihnen schwer fällt zu entscheiden, an welcher Stelle sie bei der Erstellung eines Programms beginnen sollen
- Hier könnte eventuell das sogenannte „**Eva-Prinzip**“ helfen, dass praktisch allen (interessanten) Programmen zu Grunde liegt:
  - Ein Programm beginnt mit einer **(User)-Eingabe** => Das Programm erhält also jene Daten, mit denen es arbeiten soll
  - Es folgt eine **Verarbeitung** der (Eingabe)-Daten => Zahlen-Daten könnten zur Berechnung herangezogen werden  
=> Zeichen (bzw. Texte) können analysiert oder verändert werden
  - Am Ende kommt es zu einer **(Ergebnis)-Ausgabe** => Die Rechenergebnisse oder Text-Resultate werden angezeigt

## Didaktischer Hinweis:


Wir werden zu Beginn dieses Bausteins die (interaktive) User-Eingabe noch durch eine („passive“) Variable-Zuweisung ersetzen. Die Werte der Variablen werden also noch nicht vom User bestimmt, sondern im Programm festgelegt. Diese Vorgehensweise hat den Vorteil, dass wir das EVA-Prinzip von Anfang an verfolgen können, ohne uns dabei codier-technisch zu überfordern ...

# Das Erstellen von Programmen – hilfreiche Software – Installation notwendig?

- Wir wollen im folgenden **3 Darstellungsformen** von Programmen kennenlernen, die wir gleichberechtigt trainieren werden, um umfassend auf die IHK-Abschlussprüfung vorzubereiten
- Zu deren Erstellung werden **Anwendungsprogramme** hilfreich sein. Wir empfehlen die drei folgenden: (die Sie zum Teil bereits auf Ihrem Rechner vorfinden werden)

-  „DIA“ – ein Grafiktool, das sich (unter anderem) zum Erstellen von **Programmablaufplänen** (PAP) eignet

-  „Structorizer“ – Zeichenwerkzeug zur Erstellung von **Struktogrammen** (benötigt JRE)

-  Notepad“ – Ein Texteditor, den wir für die Erstellung von **Pseudocode** verwenden

## Bemerkungen:

- Sofern Sie im Umgang mit alternativen Tools bereits geübt sind, können Sie diese selbstverständlich an Stelle der vorgeschlagenen verwenden
- Auch handschriftliche Lösungen (selbiges entspricht der Situation in der IHK-Prüfung) sind natürlich zulässig. **Voraussetzung sollte dann allerdings sein, dass Sie diese abfotografieren und hochladen können!**

# Das Erstellen von Programmen – **Musteraufgabe**

- Wir werden uns nun mit den bereits angesprochenen 3 Darstellungsformen etwas näher befassen, dabei aber zunächst nur jene (graphischen oder schriftlichen) Notationen vorstellen, die wir zur Darstellung erster Übungsaufgaben benötigen.
- Um die Verwendung der entsprechenden Symbole / Schreibweisen anschaulich einführen zu können, werden wir für alle drei Darstellungsformen jeweils eine Lösung der folgenden Musteraufgabe präsentieren:

## **Musteraufgabe:**

Das Programm startet und füllt zunächst die Variablen **a** mit dem Wert **5** und **b** mit dem Wert **7**. Anschließend wird die Summe von **a+b** berechnet und in der Variable **c** abgespeichert. Zum Schluss wird der Wert von **c** ausgegeben. Daraufhin endet das Programm

# Programmablaufplan (PAP) – Erste Symbole

- Jedes PAP startet mit einem (einzigen!) Start-Symbol:



- Und sollte mit einem einzigen End-Symbol enden:



- Ein- und Ausgaben werden als Parallelogramme notiert:



Der in den Parallelogrammen eingetragene Text teilt mit ...

- **Eingabe:** Welche Variable gefüllt werden soll
- **Ausgabe:** Von welcher Variable der Wert ausgegeben wird

# Programmablaufplan (PAP) – Erste Symbole

- Zuweisungen (von Literalen oder Rechenergebnissen) werden als **Vorgang** bezeichnet und in Rechtecken notiert:

a=5

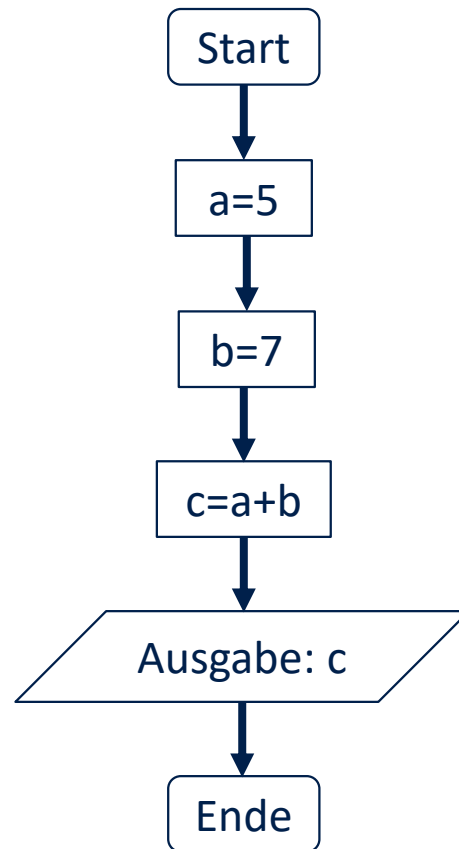
- Die **Reihenfolge** der Arbeitsschritte wird durch Pfeile dargestellt:



- Mit den (wenigen) weiteren Symbolen eines **PAP** werden wir uns erst in den nächsten Tagen beschäftigen



# Programmablaufplan (PAP) – Musterlösung



**Hinweis:** Diese klassische Darstellung verzichtet auf Farben. Es ist aber auch zulässig, die Symbole farblich zu unterscheiden.

# Struktogramme (Nassi-Shneiderman-Diagramme) – Erste Symbole

- Struktogramme werden „**von oben nach unten**“ gelesen. Daher werden keine Symbole für „Start“ und „Ende“ benötigt. Ebenso wird auf Pfeile verzichtet.

- **Eingaben, Zuweisungen** und **Ausgaben** werden alle mit einem Rechteck dargestellt:



- Mit den weiteren Symbolen eines **Struktogramms** werden wir uns erst später beschäftigen

# Struktogramme (Nassi-Shneiderman-Diagramme) – Musterlösung

a=5
b=7
c=a+b
Ausgabe: c

# Pseudocode – Erste Schreibweisen

- Pseudocode ist eine schriftliche Form des Programmierens und arbeitet mit Texten, die an einen „echten“ (Quell)-Code erinnern sollen. Der Pseudocode richtet sich an Programmierer, ohne sich jedoch dabei festzulegen, welche Programmier-Sprachen diese Programmierer bereits beherrschen. **Ziel ist es vielmehr, für möglichst jeden Programmierer verständlich zu sein.**
- Ähnlich wie beim echten Quellcode wird auf ein gewisses „**Layout**“ geachtet, um die Lesbarkeit zu verbessern. Dies bedeutet vor allem, dass sogenannte Anweisungsblöcke durch **geschweifte Klammern** – also { und } - eingerahmt werden, so dass deutlich wird, wo ein solcher Block beginnt und wo er aufhört. Ferner werden alle Anweisungen, die zu diesem Block gehören, innerhalb der geschwungenen Klammer **eingerückt** notiert. (siehe hierzu die Musterlösung)
- **Eingaben, Zuweisungen** und **Ausgaben** werden so notiert, wie wir dies bereits beim PAP und Struktogrammen kennengelernt haben

# Pseudocode – Musterlösung

Hier beginnt der Anweisungsblock

Programm „Name des Programms“ (falls bekannt)

{

a=5  
b=7  
c=a+b  
Ausgabe: c

Alle Anweisungen des Blocks werden eingerückt

Hier endet der Anweisungsblock

}

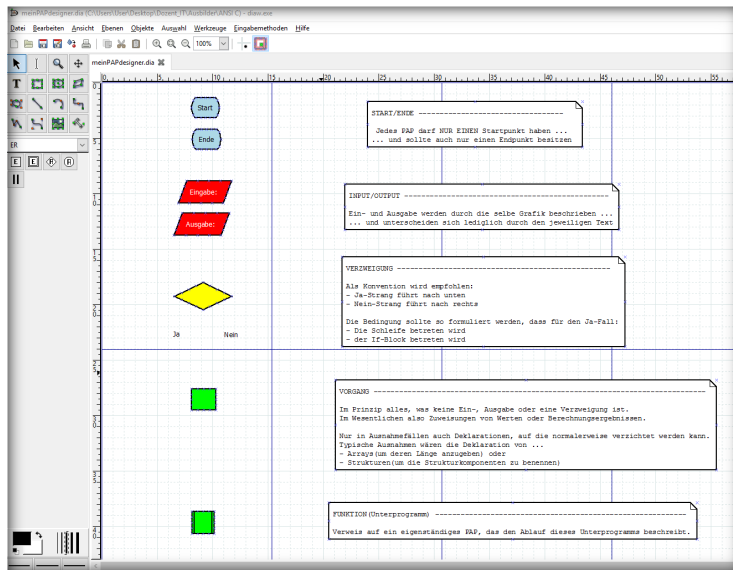
Breite der Einrückung: In der Regel 1 Tabulator-Schritt

# Das Erstellen von Programmen – Aufgabe A\_01\_01\_01

(Woche) (Tag) (Nummer)

- Wir werden nun zum Abschluss dieser Präsentation eine erste Übungsaufgabe gemeinsam bearbeiten, um insbesondere die Bedienung der Grafiktools zu trainieren.
- Für diesen Zweck wird empfohlen, die gezeigten Arbeitsschritte parallel zu deren Vorführung eigenständig mit den gewählten Anwendungsprogrammen nachzustellen.

(siehe: [Ecampus/Unterrichtsmaterialien/Tag\\_01/Unterrichtsfolien/meinPAPdesigner](#))



## Aufgabe\_01\_01\_01

Zu Beginn des Programmes wird der Variable **a** der Wert **1000** zugewiesen.

Anschließend wird in der Variable **b** das Ergebnis der Rechnung „**a geteilt durch 10**“ abgespeichert.

Daraufhin wird die Variable **c** mit dem Ergebnis der Rechnung „**b geteilt durch 10**“ gefüllt.

Schließlich wird der Variablen **d** das Ergebnis aus „**c geteilt durch 10**“ zugewiesen.

Zum Schluss werden die Werte von **a**, **b**, **c** und **d** ausgegeben und das Programm endet.

Erstellen Sie hierzu bitte ...

- einen geeigneten Programmablaufplan (PAP)
- ein geeignetes Struktogramm
- einen geeigneten Pseudocode

**VIELEN DANK  
FÜR IHRE  
AUFMERKSAMKEIT!**