# InDesign Satzautomation

**11748 Vorlesung von Gastdozenten**
**22.10., 12.11., 19.11.und 03.12.**
**10 Uhr bis 17 Uhr in Raum 171**

Kontakt: gregor.fellenz@publishingx.de



# Übersicht

## InDesign Programmierung mit JavaScript

- Einführung JavaScript für Printprodukte
- InDesign Objektmodell
- Programmierung und Programmierkonzepte

## Weiterführende Themen

- Zusammenspiel von strukturierten Daten und Satzautomation

## Die Folien zum Download:

http://www.indesignblog.com/hdm

# Literatur, Informationen und Quellen

## JavaScript

- **SelfHTML**
  http://de.selfhtml.org/javascript/
  Weblastig, JavaScript Einsteigerseite, leider nicht so gut wie der HTML Teil.
- **JavaScript: The Definitive Guide, Fifth Edition, David Flanagan**
  http://oreilly.com/catalog/9780596101992/index.html
- **JavaScript: The Good Parts, Douglas Crockford**
  http://oreilly.com/catalog/9780596517748/index.html
  Anspruchsvolle Übersicht zu Konzepten und zum Einsatz von JavaScript

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

## InDesign und ExtendScript

- **Linksammlung** unter http://www.indesignblog.com/connect/
- **InDesign automatisieren – Skripting, GREP & Co.**
  Gregor Fellenz, dpunkt verlag
  http://www.indd-skript.de
- **InDesign mit JavaScript automatisieren**
  Peter Kahrel, Deutsche Übersetzung von Martin Fischer, O'Reilly, 2007
- **Adobe Scripting Guide**
  http://www.adobe.com/products/indesign/scripting/
- **Adobe Scripting Forum** (sehr rege Community)
  http://forums.adobe.com/community/indesign/indesign_scripting
- **HilfdirSelbst Forum**
  http://www.hilfdirselbst.ch/foren/Adobe_InDesign_Forum_4.html

# Ziele – oder wozu das alles?

In jedem Layoutprozess fallen routinemäßige Aufgaben an, die sich zwar durch Handarbeit lösen lassen, aber in der Ausführung zeit- und nervenaufreibend sind. Skripte können viele dieser Aufgaben übernehmen.

- **Fertige Scripts** einsetzen oder minimal anpassen...
  Reichhaltige Auswahl von Adobe Skripts oder im Netz
- **Kleine Helferlein** wiederkehrende Aufgaben lösen.
  Statt 10 mal klicken einmal programmieren. Verweise auflösen, Sanftes spationieren
- **Dokumentanalyse**
  Preflight selber programmieren
- **Layoutautomatisierung** aus strukturierten Daten
  Fertige oder fast fertige Layouts erstellen
- Layouts aus **XML-Daten** steuern

**InDesign Satzautomation**
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

# Skripte installieren

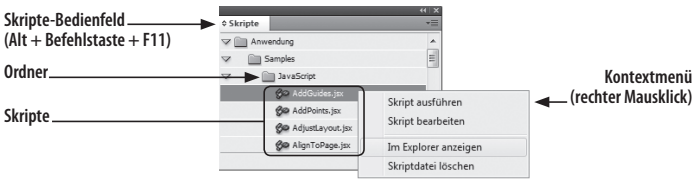
Dreh- und Angelpunkt für die Verwendung von Skripten in InDesign ist das **Bedienfeld Skripte**

- CS3/CS4: Fenster › Automatisierung › Skripten
- CS5/CS5.5: Fenster › Hilfsprogramme › Skripte

Über das Kontextmenü im Skript-Bedienfeld erreichen Sie den Ordner im Dateisystem.

Im sich öffnenden Explorer bzw. Finder navigieren Sie in den Unterordner **Scripts Panel**. In diesen Ordner kopieren Sie das Skript.

Wechseln Sie nun zurück zu InDesign. Das Skript erscheint im Bedienfeld und kann nun durch einen Doppelklick ausgeführt werden.



## Anwendungsordner

Windows XP: `Programme\Adobe\Adobe InDesign [Versionsname]\Scripts`
Windows 7: `Program Files (x86)\Adobe\Adobe InDesign [Versionsname]\Script`
Mac OS: `Programme/Adobe InDesign [Versionsname]/Scripts/Scripts Panel`

Es handelt sich um das Unterverzeichnis Scripts des Installationsordners. Der `[Versionsname]` ist je nach installierter Programmversion CS3, CS4, CS5 oder CS5.5.

## Benutzerordner

Windows XP: `Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Adobe\InDesign\Version [Version]\de_DE\Scripts\Scripts Panel`
Windows 7: `Benutzer\[Benutzername]\AppData\Roaming\Adobe\InDesign\[Version]\de_DE\Scripts\Scripts Panel`
Mac OS: `Benutzer/[Benutzername]/Library/Preferences/Adobe InDesign/[Version]/de_DE/Scripts/Scripts Panel`

Der `[Benutzername]` entspricht dem Systembenutzernamen, die `[Version]` ist 5.0 für CS3, 6.0 für CS4 und 7.0 für CS5. Der Ordner für die Lokalisierung (de_DE) kann je nach installierter Programmsprache unterschiedlich sein, in der Version CS3 entfällt der Unterordner für die Lokalisierung.

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

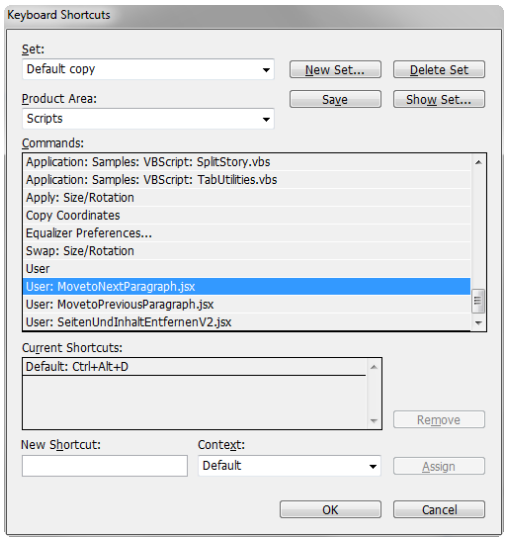
# Beispielskripte

## Einfach aber effektiv

- `01_MoveParagraphScripts`
  Jongware *http://bit.ly/osvJLI*
  Absätze verschieben
- `01_TransposeTwoCharacters.jsx`
  Gilbert Consulting *http://bit.ly/9MclcI*
  Buchstaben vertauschen

## Noch effektiver: Tastaturkürzel für Skripte festlegen

- Bearbeiten › Tastaturbefehle...
- Produktbereich › Skripten
- Kürzel eingeben **und** Zuweisen drücken



**Scripting in InDesign**: VB Skript, AppleScript und JavaScript

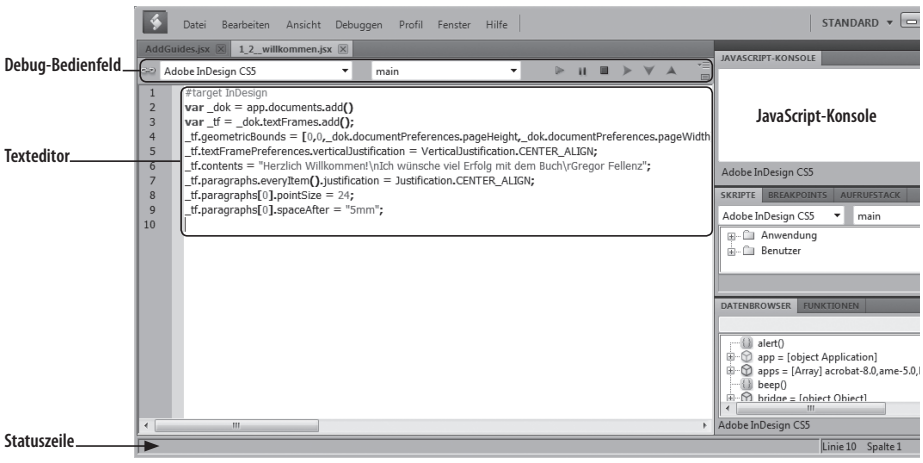
# Einführung JavaScript


- **Plattformunabhängig** (Mac/Windows/[Linux])
- Sprache ist aus der **Web-Programmierung** bekannt.
  Clienstseitige Schnittstelle (API) für den Zugriff auf HTML-Dokumente
  Browser Sandbox aus Sicherheitsgründen
- Kombination aus **Sprachkern** und **Document Object Model** (DOM)
  Wir lernen beides zusammen, der Sprachkern ist auch im Web einsetzbar.
- **Leichte** und **schlanke** Sprache
  Wer schon mal programmiert hat lernt JS schnell.
- InDesign, Creative Suite eigentlich **ExtendScript**
- Bei InDesign meist prozedurale Programmierung – es wird aber auf Objekte zugegriffen.
  Layouts/Bücher sind prozedural, haben eine definierte Abfolge und Ende.
- ExtendScript kann auch auf das **Dateisystem** zugreifen
  Adobe Implementierung

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

## Werkzeuge und Hilfsmittel

- **Texteditor** ausreichend
- **Adobe ExtendedScript Toolkit**
  Editor/Mini **IDE** (Integrierte Entwicklungsumgebung) für InDesign
  Ideal zum Entwickeln von Skripten, Syntaxhervorhebung
- Befehlszeile, **Konsole** zum testen
- Debugging **Einzelschritte** und **Breakpoints**
- Hilfe mit Objektmodell



## Kann's endlich mal losgehen?

JavaScript-Kochrezept:

1. **InDesign** und **Adobe ExtendScript** starten
2. Programm schreiben
3. Entweder direkt aus der IDE oder in der InDesing Skriptpalette ausführen
4. Fertig!

- **Speicherort**: Über das Skript Panel am einfachsten zu finden.
- **Musterskript** `01_helloWorld-1.jsx` im Ordner `01_termin`

```
var _dokument = app.documents.add();
var _tf = _dokument.textFrames.add();
_tf.geometricBounds = [20,20,100,100];
_tf.contents = „hallo welt";
```

InDesign Satzautomation
Übungsaufgabe

## Extended Script Toolkit lieben lernen...

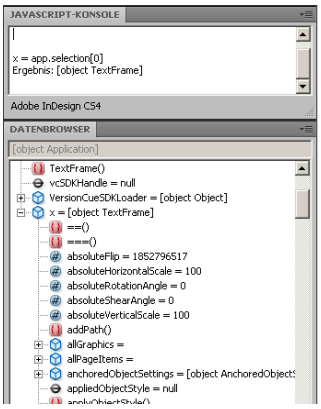
- Öffnen Sie das **Extended Script Toolkit**
- Öffnen Sie die Datei `02_helloWorld-2.jsx`
- Öffnen Sie die Datei `02_ESTK.indd` in InDesign

! Erkunden Sie die Bedienoberfläche

! Setzen Sie einen Breakpoint
Verwenden Sie den Einzelschritt Modus

! Erstellen Sie ein neues Skript (STRG + N) schreiben `app.selection[0]` und wählen Ausführen aus dem Debugmenü. Dieses Zeile zeigt, was im InDesign-Dokument ausgewählt ist. In der JavaScript-Konsole zeigt ESTK `[objekt InsertionPoint]` an.

! Schreiben Sie `app.selection[0]` direkt in die Konsole!



## Syntax – wie schreibt man das eigentlich?

Der **Syntax** beschreibt den Aufbau, die Grammatik und auch die „Rechtschreibung“ einer Programmiersprache. JavaScript ist sehr tolerant, trotzdem sollte man sich von vornherein einen guten Stil angewöhnen. Komplexere Skripte werden sonst schnell sehr unleserlich.

- Skripte in Ausführungsreihenfolge, von „oben nach unten“.
- JavaScript ist **case-sensitive**, Namen immer gleich schreiben! `abc` ist nicht `Abc`,
  Namen können nicht doppelt vergeben werden. Keine Leer- oder Sonderzeichen in Namen!
- Es gibt reservierte Wörter (im ESTK fett+blau) z.B.: `if`, `else`, `false`, `for`, `null`, `true`, `var`, ...
- **Text** muss immer mit Anführungszeichen (`"..."` oder `'...'`) umschlossen sein.
- **Zahlen** ohne Anführungszeichen (einfach `5` oder `85`), Kommazahlen mit Punkt (`3.45`).
- **Semikolon** wird als Trennzeichen zweier Befehle verwendet `;`
  Zeilenschaltung funktioniert, aber nicht immer eindeutig
  Tabs, Leerzeichen, Return = Whitespace
- Kommentare werden nicht ausgeführt und können mit `//` eingeleitet werden.

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

## Klammern

- Runde Klammern `()` werden für Bedingungen in if-Abfragen, für Schleifendefinitionen, für die Parameterübergabe von Methoden und Funktionen sowie innerhalb von Berechnungen von Zahlen verwendet.
- Eckige Klammern `[]` werden für den Zugriff auf Sammlungen und Arrays sowie für die Definition von Arrays verwendet.
- Geschweifte Klammern `{}` umschließen die Codeblöcke von Abfragen, Schleifen und selbst entwickelten Funktionen.

## Variablen

- Eine Variable ist ein **Platzhalter**, Speicherplatz, Referenz, „Lesezeichen“
- Der **Wert** einer Variablen kann sich während des Ablaufs **ändern**
  An verschiedenen Stellen des Programms kann sie ganz unterschiedliche Werte haben.
- Variablen sollten durch das Schlüsselwort **var** deklariert werden
- Einer Variable kann ein Wert zugewiesen werden, dazu dient der **Zuweisungsoperator** =
- `var _variablenName = Wert`
- Case-sensitiv, d.h. **zahl** und **Zahl** sind zwei verschiedene Variablen
  Tipp: Vermeiden Sie Umlaute und Sonderzeichen in Variablennamen.
- Kann beliebige Werte aufnehmen, übliche Unterscheidungen:
  - **Zahlen** (Kommazahlen immer mit Punkt abtrennen z. B. 12.05)
  - **Strings** (Texte in Anführungszeichen z. B. "ein Text")
  - **Boolesche Werte** (Wahr oder falsch, true bzw. false),
  - **Objekte** (Ein InDesign Dokument, Array)

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

## Arithmetische Operatoren

JavaScript kann rechnen, dazu stehen die folgenden **Rechenoperatoren** zur Verfügung:

| | |
|---|---|
| + | Addition (und Verbinden von Strings) |
| - | Subtraktion |
| * | Multiplikation |
| / | Division |
| % | Modulo (Restrechnung) |
| ++ | Erhöhung (Inkrement) |
| -- | Minderung (Dekrement) |

## Mit Variablen umgehen

- Öffnen Sie die Datei `02_helloWorld-2.jsx` im Extended Script Toolkit

! Verwenden Sie für den Text `"hallo Welt"` eine Variable.

! Verwenden Sie die JavaScript Konsole um Variablen zu analysieren.

! Durch den **Zuweisungsoperator** „=“ wird der Ausdruck rechts vom Gleichheitszeichen ausgewertet und dann dem links stehenden Ausdruck zugewiesen (In diesem Fall liest man also von rechts nach links!).
**Berechnen** Sie die Summe zweier Zahlen und geben Sie das Ergebnis im Textrahmen aus.

! Beachten Sie: Die Eigenschaft `contents` kann nur Zeichenketten aufnehmen.
Eine Zahl kann einfach in eine Zeichenkette konvertiert werden. „Addieren“ Sie dazu eine leere Zeichenkette: `5 + ""`

! Was passiert wenn Sie Text und Zahl addieren?



## Texte

Text muss innerhalb von JavaScript in Anführungszeichen "" stehen. Texte werden in der Informatik **Strings** genannt.

- Ein String ist eine Zeichenkette, also ein Folge von zusammen gespeicherten Zeichen.
  `var _text = "Viele Zeichen ergeben einen Text"`
- Steht ein + zwischen zwei Strings werden diese dadurch zu einem Gesamtstring verbunden.
- Sonderzeichen in Strings werden mit dem **Escape Zeichen** eingeleitet

  `\n` **Neue Zeile** Soft Return

  `\r` **Neue Zeile** Return

  `\"` **Anführungszeichen** für die Verwendung von Anführungszeichen innerhalb des Texts

  `\'` **Einfaches Anführungszeichen**

  `\\` **Backslash.** Um einen \ auszugeben muss z. B. „c:\\pfad\\datei.txt“ geschrieben werden.
  Ein einfacher Backslash wird als Beginn eines Steuerzeichens interpretiert.

## Zeichenketten

- Öffnen Sie die Datei `02_helloWorld-2.jsx` im Extended Script Toolkit

! Strings können mit dem + Operator verknüpft werden.
Fügen Sie die Strings `"hallo"` und `"welt!"` zusammen.

! Fügen Sie einen String mit Zeilenschaltung zusammen.

- Zahlen und Strings mischen

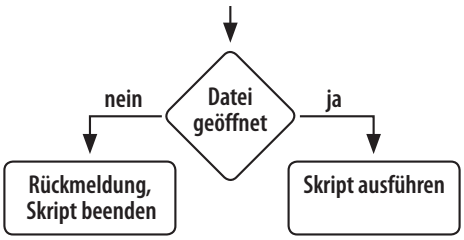
! Wenn `var a` den String `"4"` speichert und `var b` den String `"9"` ist das Ergebnis der Addition der Strings `"49"` nicht die Zahl 13.
Mit der Funktion `Number("123")` können Strings in Zahlen verwandelt werden,
Mit der Funktion `(123).toString()` können Zahlen in Strings verwandelt werden
Korrigieren Sie `03_Casting.jsx`



## Anweisungen durch Abfragen

Während des Programmablaufs können **Abfragen** ausgewertet werden. Dazu wird eine Kontrollstruktur benötigt, die zur Formulierung von Bedingungen dient. Damit können **Verzweigungen** im Ablauf abgebildet werden.

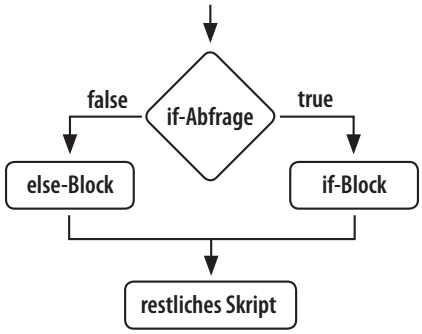
- Bedingte **if-Anweisungen** werden verwendet, um einen Zustand zu prüfen




Code Beispiel Prüfung:

```
if ( app.selection.length == 0 ) {
  alert("Es ist nichts ausgewählt");
}
```

- Klassisch ist die Abfrage **If else** mit zwei Anweisungsblöcken
  Wenn eine Bedingung **zutrifft** wird der erste Anweisungsblock ausgeführt (if)
  Wenn die Bedingung **nicht zutifft** wird der zweite Anweisungsblock ausgeführt (else)




Code Beispiel Verzweigung

```
if ( app.selection.length == 0 ) { ... }
else {
  alert("Es ist etwas ausgewählt");
}
```

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

## Wahr oder falsch?

Die Werte **true** und **false** kommen aus der Aussagenlogik. Die Werte sind logischerweise jeweils immer genau richtig oder immer falsch.

- In JavaScript sind die Bezeichner `true` und `false` reserviert, d.h. sie dürfen nur im eigentlichen Sinne verwendet werden und z. B. nicht als Variablennamen.
- Die if Abfrage testet immer, ob der **Ausdruck** (Vergleich/Frage) mit true beantwortet wurde.
  `if (true) { //wird immer ausgeführt }`
- true entspricht dem Zahlenwert 1
  false entspricht dem Zahlenwert 0
  Das verwirrende Beispiel:
  `false == 0 // Ergebnis: true`
- Der zugehörige Datentyp heißt **Boolean** und kann auch in Variablen verwendet werden.

## Vergleichsoperatoren für Abfragen

Es gibt sechs **Vergleichsoperatoren** um **Bedingungen** (Ausdrücke) zu formulieren:

- `==` gleich **Achtung:** Ein Zuweisung **x=11** wird mit einem = vorgenommen. Der Vergleichsoperator für Abfragen **if(x == 11)** besteht aber aus zwei `==` .
- `!=` ungleich
- `>` größer
- `<` kleiner
- `>=` größer oder gleich
- `<=` kleiner oder gleich

## Logische Operatoren

Komplexe Abfragen können mit **logischen Operatoren** verknüpft werden:

- `&&` bedeutet logisch und
- `||` bedeutet logisch oder
- `!` bedeutet nicht (Umkehrung)

InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

| Vergleichs-operator | Beschreibung: Der Vergleich trifft zu, wenn … | Beispiel für einen zutreffenden Vergleich | Beispiel für einen nicht zutreffenden Vergleich |
|---|---|---|---|
| `==` | beide Werte gleich bzw. identisch sind. | `1 == 1`<br>`"Text" == "Text"` | `1 == 2`<br>`"Text" == "2. Text"` |
| `!=` | beide Werte ungleich sind. | `1 != 2`<br>`"Text" != "2. Text"` | `1 != 1`<br>`"Text" != "Text"` |
| `<` | der linke Wert kleiner als der rechts vom Operator ist. | `1 < 2` | `100 < 5` |
| `<=` | der linke Wert kleiner oder gleich dem rechts vom Operator ist. | `1 <= 1` | `100 < 99.9` |
| `>` | der linke Wert größer als der rechts vom Operator ist. | `2 > 1` | `5 > 100` |
| `>=` | der linke Wert größer oder gleich dem rechts vom Operator ist. | `2 >= 2` | `99.9 >= 100` |

## Abfragen

- Öffnen Sie die Datei `04_Abfragen.jsx` im Extended Script Toolkit

! Die Variable `_seitenAnzahl` ist mit der Länge der Seiten im aktuell geöffneten Dokument belegt. Geben Sie alternative Texte in Abhängigkeit von der Dokumentlänge aus.
Testen Sie Ihre Kontrollstruktur, indem Sie dem Dokument Seiten hinzufügen.

! Spielen Sie mit den Vergleichoperatoren. Fragen Sie z. B. eine Mindestseitenanzahl ab.

! Testen Sie zwei Texte auf Gleichheit – gibt es einen Unterschied?
Was testet der „>“ bzw. „<“ Vergleichsoperator wenn er auf Strings angewendet wird?

! Definieren Sie Variablen mit den reservierten Wörtern `true` und `false` und verwenden Sie diese in Abfragen.

! Mit dem Vergleichsoperator == können auch Objektzustände abgefragt werden.
Wenn keine Dokument geöffnet ist, enthält das Objekt `app.documents[0]` den Wert **null**.
Der „Wert“ `null` ist von der Ziffer 0 oder einem leeren String („“) zu unterscheiden.
Er hat die Bedeutung **leer**, **unbestimmt**, „ohne Wert“

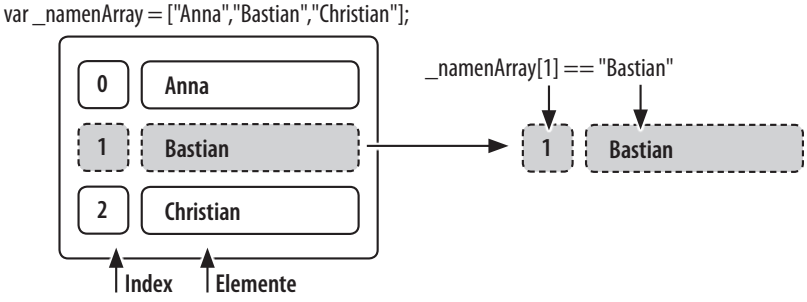
! Prüfen Sie, ob ein Dokument geöffnet ist.



## Arrays

Arrays sind eine häufig verwendete Datenstruktur in JavaScript. Mit **Arrays** können Listen von Zahlen oder Zeichenketten gesammelt/zusammen verwaltet werden.

- Ein Array ist eine Liste von Einzelelementen
- Definition über Variablendeklaration:
```
var _namenArray = ["Anna", "Bastian", "Christian"];
```
- Oder als leeres Array
```
var _namen = [];
```



- Anhängen von neuen Elementen
  `_namen.push("Tobias");`
- **Zählung beginnt bei 0**
- Elemente eines Arrays werden die Position (index) in eckigen Klammern adressiert.
  Das Ergbnis von `_namen[0]` ist `"Markus"` von `_namen[2]` ist `"Thomas"`
- Arrays haben viele nützliche Funktionen/Eigenschaften
  Die Länge eine Arrays (d.h. die Zahl der Elemente): `_namen.length`
  Arrays sortieren: `_namen.sort();`
- Ein Array kann mit `join()` zu einem String zusammengeführt werden.
- Strings können mit `split("TRENNER")` in Arrays überführt werden

**InDesign Satzautomation**
Übungsaufgabe

## Arrays

- Öffnen Sie die Datei `05_Anordnungen.jsx` im Extended Script Toolkit

! Ermitteln Sie die Länge des Arrays und geben Sie diese aus.

! Fügen Sie ein neues Element hinzu! Prüfen Sie das Ergebnis.

! Geben Sie die Inhalte des Arrays aus.

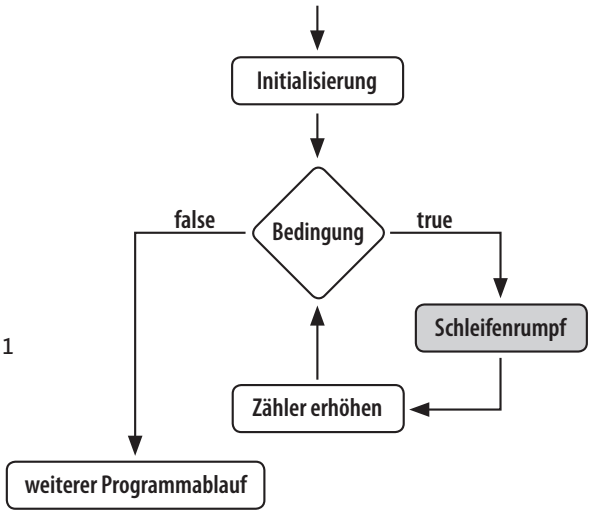
! Sortieren Sie den Array und geben Sie die Inhalte aus. Testen Sie die Sortierung mit deutschen Umlauten.

! Eine Sonderform ist der **assoziative Array**. Es verwendet keinen numerischen Index, sondern sogenannte Schlüssel zur Indizierung und damit zur Adressierung der Elemente (Wörterbuch).
In JavaScript kann anstatt eines Zahlenindex ein String verwendet werden.
Erstellen Sie einen assoziativen Array!

## Schleifen

Eine Schleife ist eine Kontrollstruktur, mit der man eine Gruppe von Anweisungen mit einer bestimmten Anzahl von Wiederholungen ausführen kann.

- Sehr oft gebraucht und einfach sind **for-Schleifen**, es gibt noch **while-Schleifen**
  - **Datenstruktur:**
    Array
  - **Startwert der Zählvariable:**
    der Index des ersten Elements
  - **Ausführungsbedingung:**
    solange noch Elemente im Array vorhanden sind
  - **Schrittweite:**
    da jedes Element besucht werden soll 1
  - **Schleifenrumpf:**
    Ausgabe des Elements in einem Hinweisfenster




InDesign Satzautomation
Übersicht | JavaScript | InDesign Objektmodell | InDesign Programmierung | InDesign XML

Code-Beispiel for Schleife:

```
for (var i = 0; i < _array.length; i++) {
  //Anweisungsblock
}
```

- **Arrays** (und Sammlungen) eignen sich perfekt für Schleifen:
  Arrays haben einen Startwert (Es wird von 0 gezählt)
  Arrays haben eine Endbedinung, die Länge (Eigenschaft `length`) die abgearbeitet werden muss

## Schleifen Übung

- Öffnen Sie die Datei `06_Schleifen.jsx` im Extended Script Toolkit

! Schreiben Sie eine Funktion zum nummerieren von Seiten
Die Sammlung (ähnlich einem Array) von Seiten erhalten Sie mit `app.documents[0].pages;`

! Bauen Sie einen Text aus aktueller Seitenzahl und Gesamtseitenlänge in der Form Seite 1 von 10.

! Legen Sie einen Array mit vier Namen an. Fügen Sie den Array in einem Textrahmen zusammen ohne die Funktion join() zu verwenden.

! Platzieren Sie 5 Textrahmen mit beliebigem Inhalt auf einer Seite!