JavaScript

Arber Osmani

15. Dezember 2021

Inhalt

Was ist JavaScript?

JavaScript ist eine *leichtgewichtige*, *interpretierte*, *prototypenbasierte*, *multiparadigma* Programmiersprache mit *first-class functions*.

JavaScript unterstützt objektorientierte, prozedurale und funktionale Programmierung.

Was ist JavaScript?

JavaScript ist eine *leichtgewichtige*, *interpretierte*, *prototypenbasierte*, *multiparadigma* Programmiersprache mit *first-class functions*.

JavaScript unterstützt objektorientierte, prozedurale und funktionale Programmierung.

Was ist JavaScript?

ECMAScript

JavaScript implementiert die Sprachspezifikation von **ECMAScript**[1]. Herausgeber ist ECMA International mit Sitz in Genf. Ziel ist, die Sicherstellung der Interoperabilität von Webseiten über verschiedene Webbrowser hinweg.

[1] ECMA-262

Was ist JavaScript?

Basisdaten

- Erschienen im Jahr 1995
- Aktuelle Version ist *ECMAScript 2021*
- Dynamische und schwache Typisierung
- Bekannte Implementierungen sind *SpiderMonkey*, *JavaScriptCode* und *V8*
- Einsatz im Frontend und Backend möglich

Eigenschaften

Typisierung

JavaScript definiert 7 Datentypen

Primitive Datentypen

- Null
- Undefined
- Boolean
- Number
- String
- Symbol

Komplexe Datentypen

- Objekte

Eigenschaften

Typisierung

Der Datentyp einer Variable wird nicht explizit deklariert. Eine Variable kann mit Werten eines beliebigen Types beschrieben und überschrieben werden.

```
var foo = 123;        typeof(foo);        // number

foo = 'hallo';        typeof(foo);        // string
```

Eigenschaften

Typisierung

Der Datentyp einer Variable wird nicht explizit deklariert. Eine Variable kann mit Werten eines beliebigen Types beschrieben und überschrieben werden.

```
var foo = 123;        typeof(foo);      // number

foo = 'hallo';        typeof(foo);      // string

foo = {};             typeof(foo);      // object

foo = true;           typeof(foo);      // boolean
```

Eigenschaften

Typisierung

Der Datentyp einer Variable wird nicht explizit deklariert. Eine Variable kann mit Werten eines beliebigen Types beschrieben und überschrieben werden.

```
var foo = 123;        typeof(foo);        // number

foo = 'hallo';        typeof(foo);        // string

foo = {};             typeof(foo);        // object

foo = true;           typeof(foo);        // boolean
```

Eigenschaften

First-class functions

Funktionen werden als gewöhnliche Variable behandelt

```
let add = function(a, b) { return a + b; };

let operate = function(operation, ...operands) {
  return operation(...operands);
};
```

Eigenschaften

First-class functions

Funktionen werden als gewöhnliche Variable behandelt

```
let add = function(a, b) { return a + b; };

let operate = function(operation, ...operands) {
  return operation(...operands);
};


operate(add, 12, 8);  // 20
```

Eigenschaften

First-class functions

Funktionen werden als gewöhnliche Variable behandelt

```
let add = function(a, b) { return a + b; };

let operate = function(operation, ...operands) {
  return operation(...operands);
};


operate(add, 12, 8);  // 20
```

Eigenschaften

First-class functions

Daraus folgen auch *Anonymous function* und *Higher order functions*.

```
operate(function(a, b) { return a * b; }, 12, 8)
```

```
let multiplier = n => x => n * x;
```

Seit ES6 gibt es auch eine Kurzschreibweise mit Arrow functions[2].

```
operate((a, b) => a * b, 12, 8);  // 96
```

[2] Pfeilfunktionen haben nebst der Syntax weitere Differenzen https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Eigenschaften

First-class functions

Daraus folgen auch *Anonymous function* und *Higher order functions*.

```
operate(function(a, b) { return a * b; }, 12, 8)
```

```
let multiplier = n => x => n * x;
```

Seit ES6 gibt es auch eine Kurzschreibweise mit Arrow functions[2].

```
operate((a, b) => a * b, 12, 8);  // 96
```

[2] Pfeilfunktionen haben nebst der Syntax weitere Differenzen https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Eigenschaften

Objekte

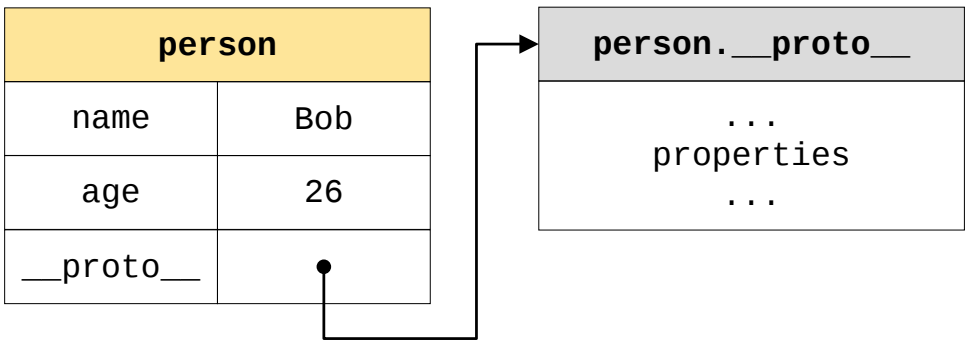
- In JavaScript ist fast alles ein Objekt [3]
- Funktionen sind auch Objekte
- Ein Objekt ist eine Kollektion aus Key-Value Paaren mit einem einzigen Prototyp Objekt (oder keinem (`null`))
- Es können auch eigene Objekte definiert werden

[3] Ausser primitiven wie `null`, `undefined`, Zeichenketten, Zahlen und Boolean Werte

Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};
```

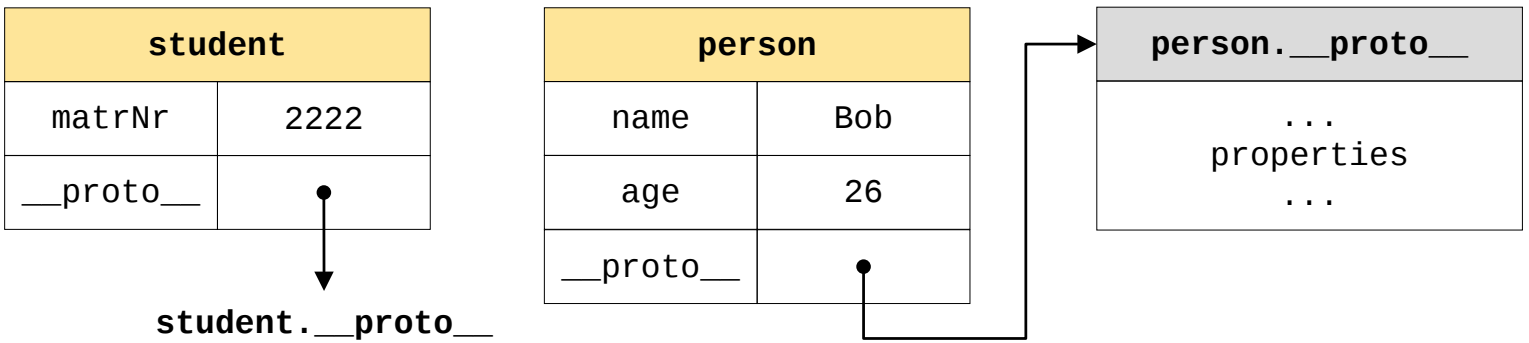


Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};
```

```
student = {
  matrNr: 2222
}
```

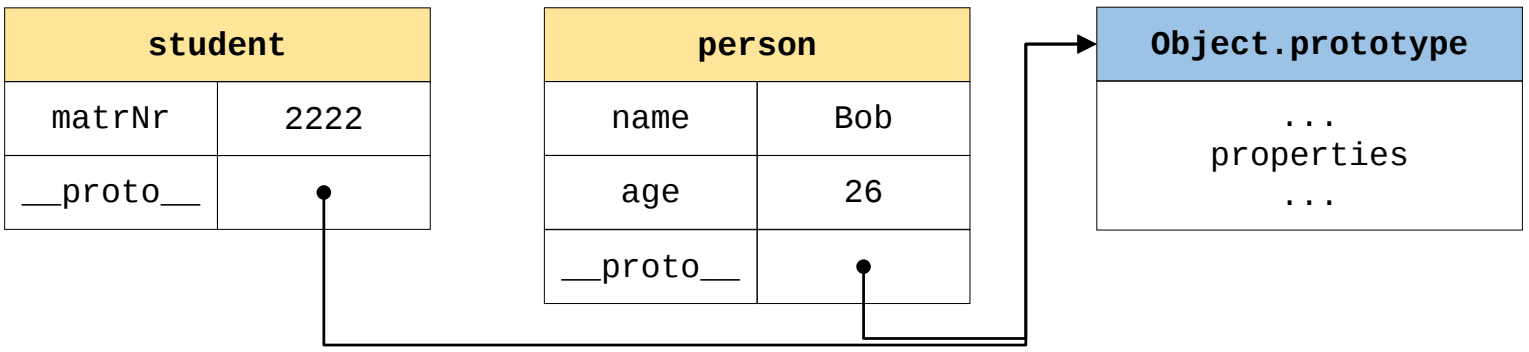


Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};
```

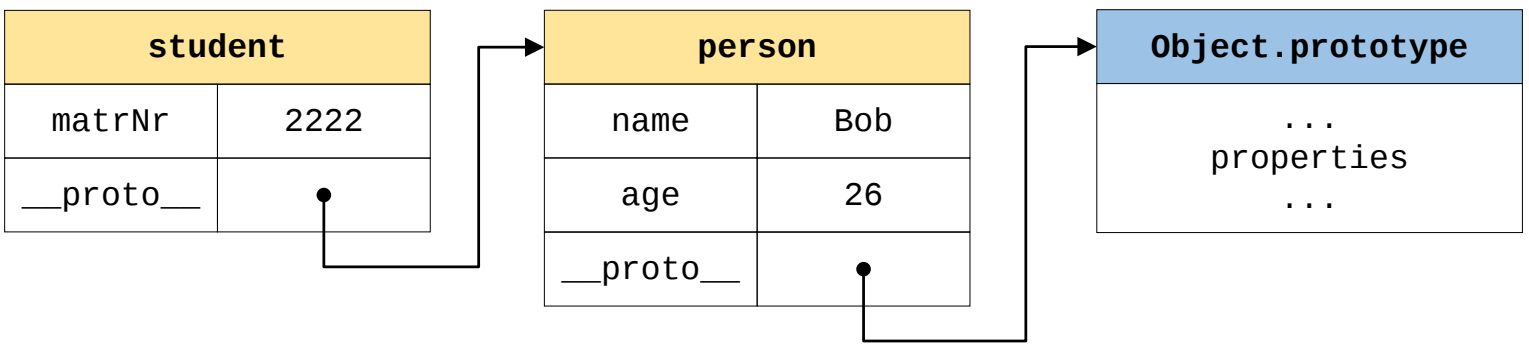
```
student = {
  matrNr: 2222
}
```



Eigenschaften

Objekte

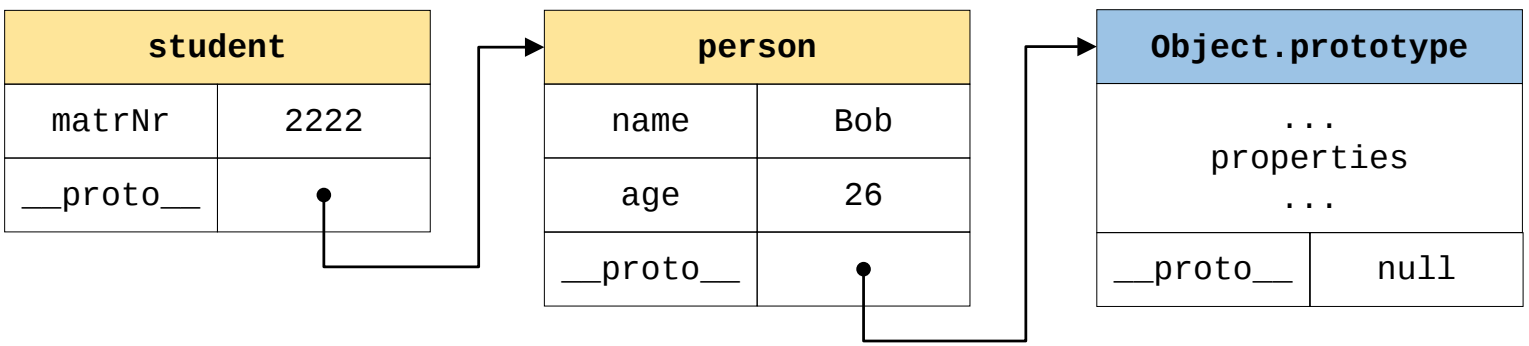
```
person = {
  name: "Bob",
  age: 26
};

student = {
  matrNr: 2222
}

student.__proto__ = person;
```



Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};

student = {
  matrNr: 2222
}

student.__proto__ = person;
```



Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};

student = {
  matrNr: 2222
}

student.__proto__ = person;
```

Diese Prototypen Verkettung wird als *Prototype Chain* bezeichnet.

Eigenschaften

Objekte

```
person = {
  name: "Bob",
  age: 26
};
```

```
student = {
  matrNr: 2222
}
```

```
student.__proto__ = person;  // DEPRECATED !!!
Object.setPrototypeOf(student, person)
student.name === 'Bob'  // true
```

Diese Prototypen Verkettung wird als *Prototype Chain* bezeichnet.

Programmierparadigma

Programmierparadigma

Objektorientierte Programmierung

- JavaScript hat keine Klassen wie Java
- Das instanziieren von Objekten geschieht via *Function constructors*
- Function constructors sind normale Funktionen, deren `this` Objekt[4] auf ein neues Objekt zeigt und dieses zurückgibt
- Dabei wird (wie bei Java) das Schlüsselwort `new` verwendet[5]
- Dabei wird die Prototype Chain automatisch erstellt

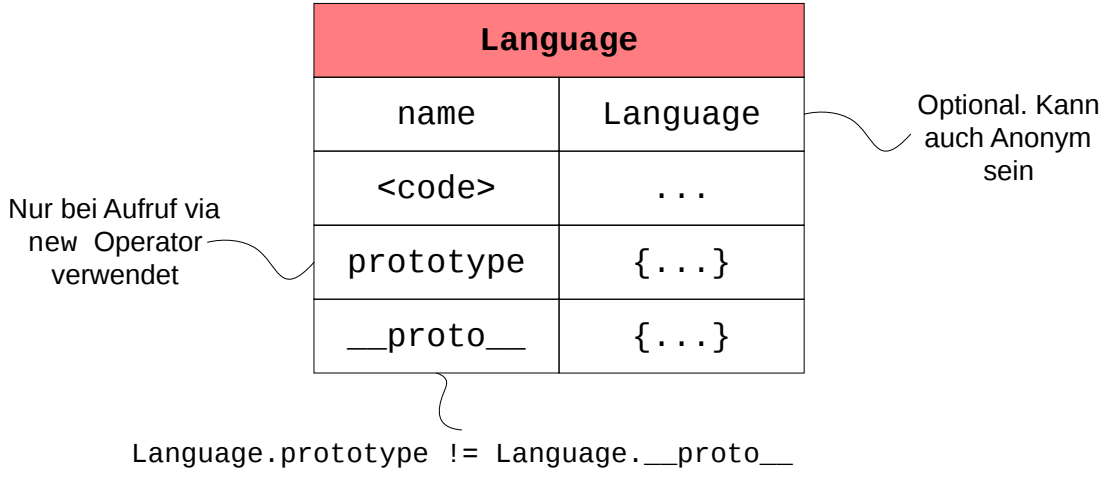
[4] Property vom *Execution Context*

[5] Eine Marketing Entscheidung `https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript`

Programmierparadigma

Objektorientierte Programmierung

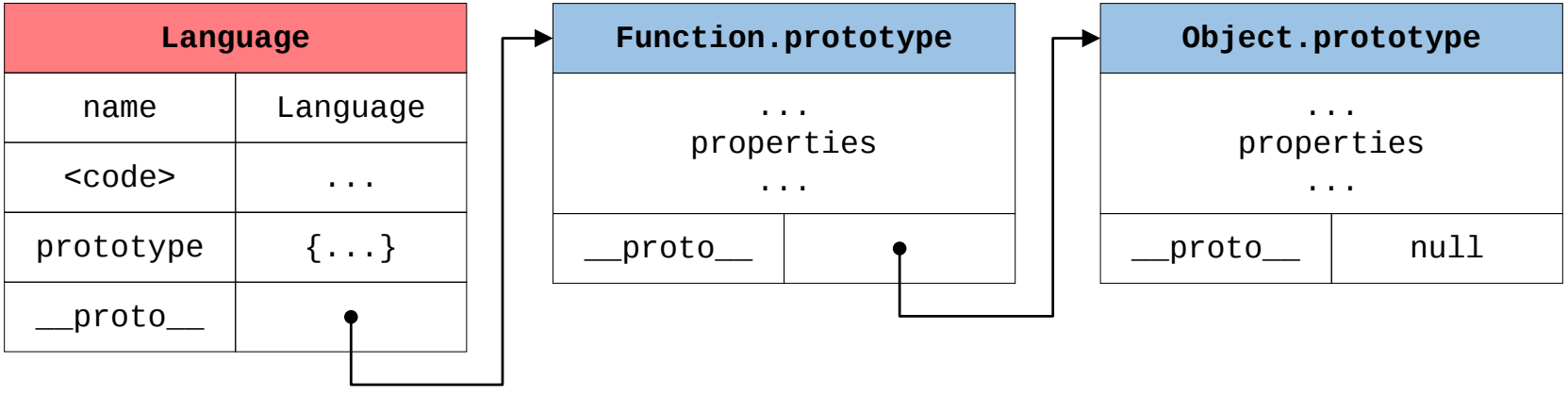
```
function Language(appeard, typing) {
  this.appeard = appeard;
  this.typing = typing;
}
```



Programmierparadigma

Objektorientierte Programmierung

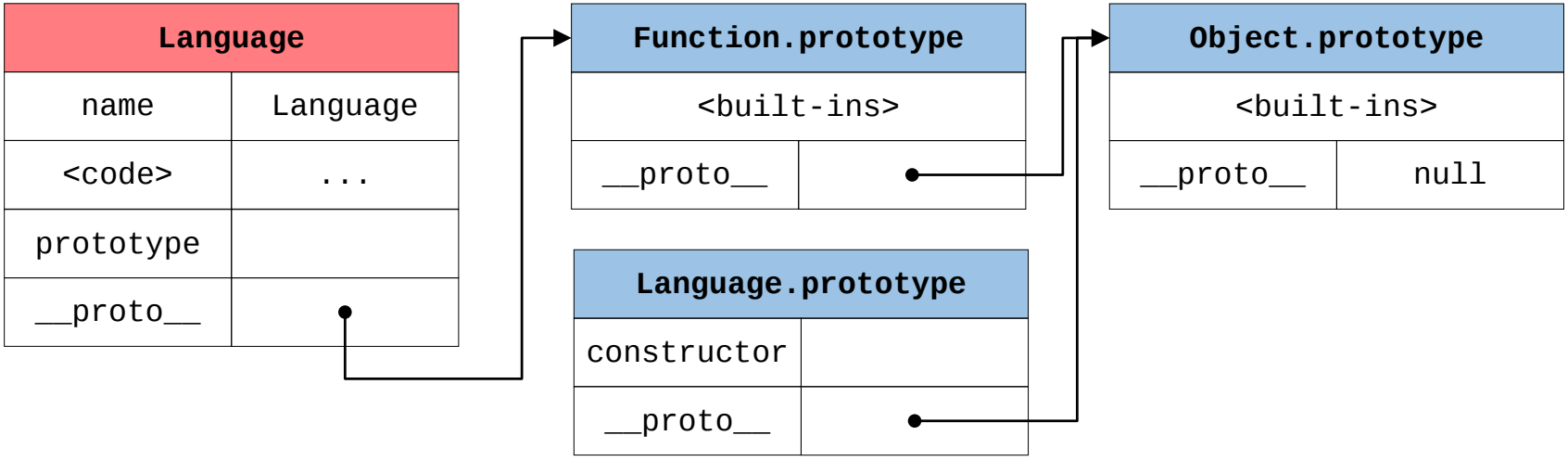
```
function Language(appeard, typing) {
  this.appeard = appeard;
  this.typing = typing;
}
```



Programmierparadigma

Objektorientierte Programmierung

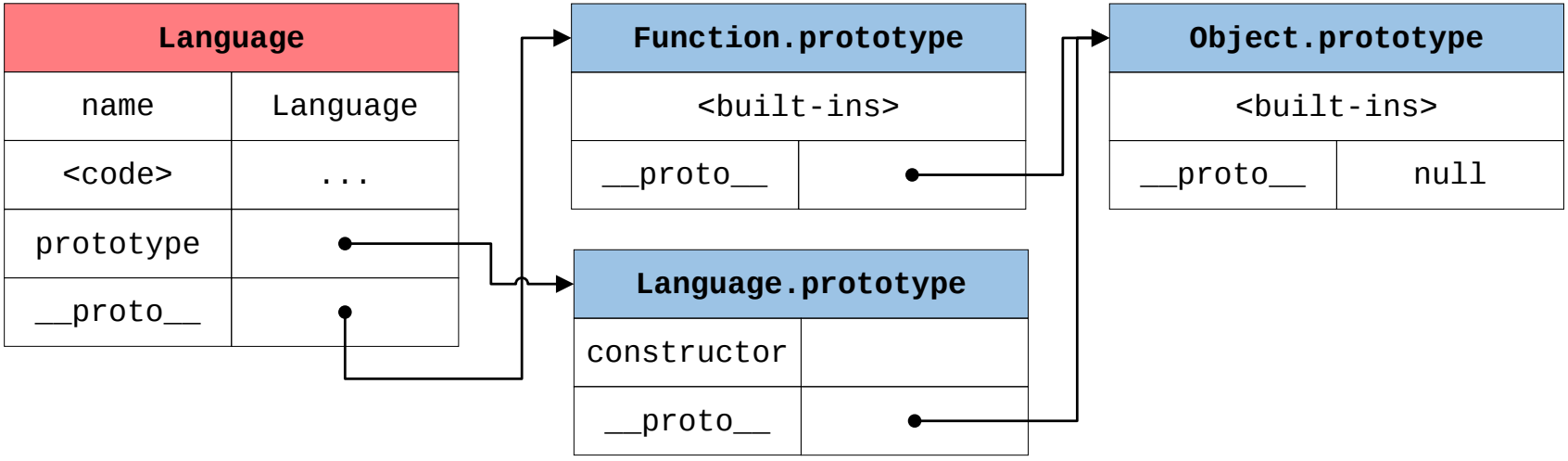
```
function Language(appeard, typing) {
  this.appeard = appeard;
  this.typing = typing;
}
```



Programmierparadigma

Objektorientierte Programmierung

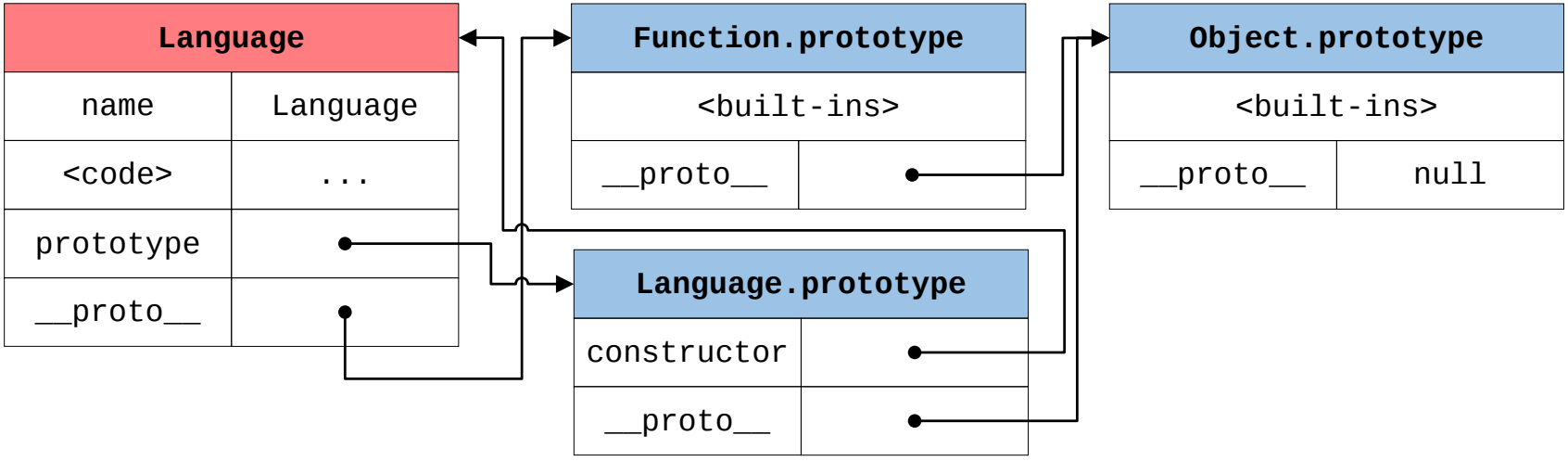
```
function Language(appeard, typing) {
  this.appeard = appeard;
  this.typing = typing;
}
```



Programmierparadigma

Objektorientierte Programmierung

```
function Language(appeard, typing) {
  this.appeard = appeard;
  this.typing = typing;
}
```



Programmierparadigma

Objektorientierte Programmierung

```
javascript = new Language(1995, 'weak')
```

Das verwenden vom `new` führt im wesentlichen die folgenden Schritte aus:

1. Erstellt ein neues Objekt `obj = {}`
2. Invoziert die Funktion `Language` mit dem Kontext von `obj`; dabei erhält `obj` die Eingenschaften `appeard` und `typing`
3. Setzt den Prototyp von `obj` zu `Function.prototype`
4. Liefert das Objekt `obj` zurück

Programmierparadigma

Objektorientierte Programmierung

```
javascript = new Language(1995, 'weak')
```

Das verwenden vom `new` führt im wesentlichen die folgenden Schritte aus:

1. Erstellt ein neues Objekt `obj = {}`
2. Invoziert die Funktion `Language` mit dem Kontext von `obj`; dabei erhält `obj` die Eingenschaften `appeard` und `typing`
3. Setzt den Prototyp von `obj` zu `Function.prototype`
4. Liefert das Objekt `obj` zurück

Programmierparadigma

Objektorientierte Programmierung

```
javascript = new Language(1995, 'weak')
```

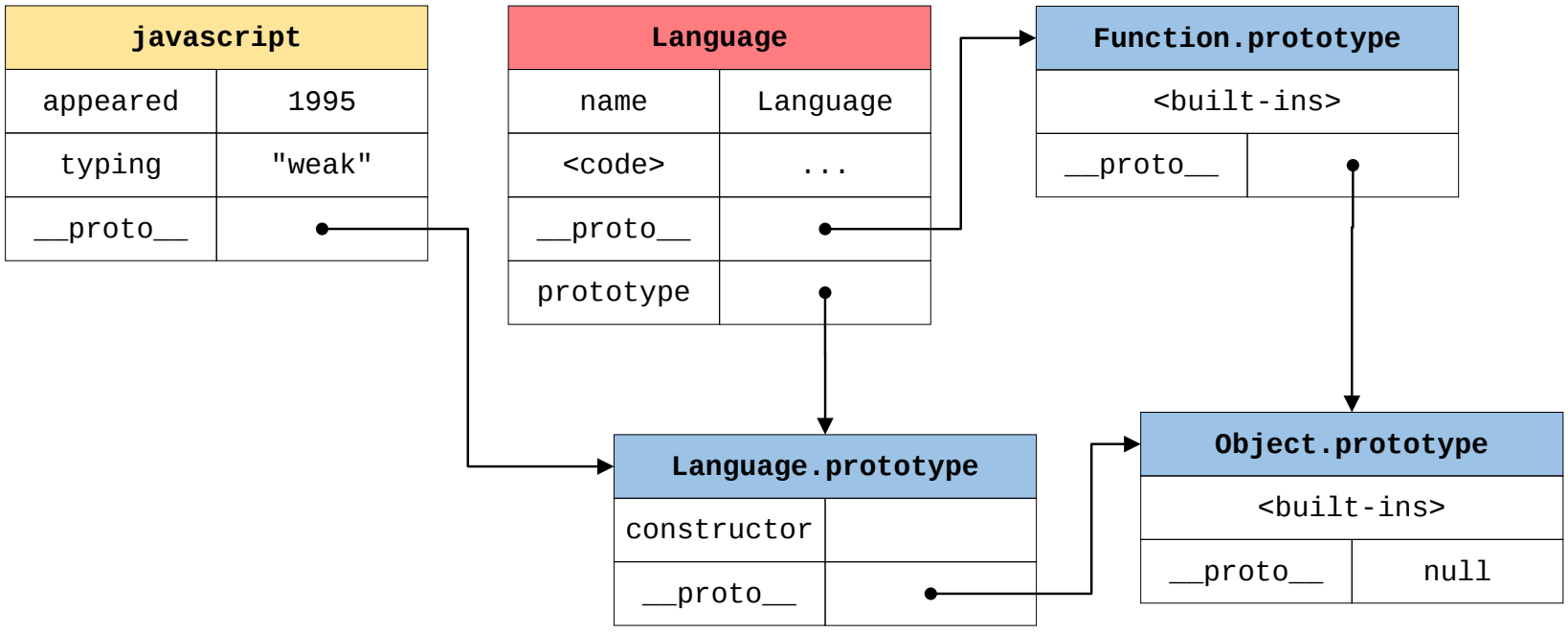
Die Schritte manuell ausgeführt:

```
// 1. Neues leeres Objekt erstellen
let obj = {}

// 2. Language mit Kontext von obj aufrufen
Language.call(obj, 1995, 'weak')

// 3. Prototyp zu Function.prototype setzen
Object.setPropertyOf(obj, Language.prototype)
```

Programmierparadigma

Objektorientierte Programmierung

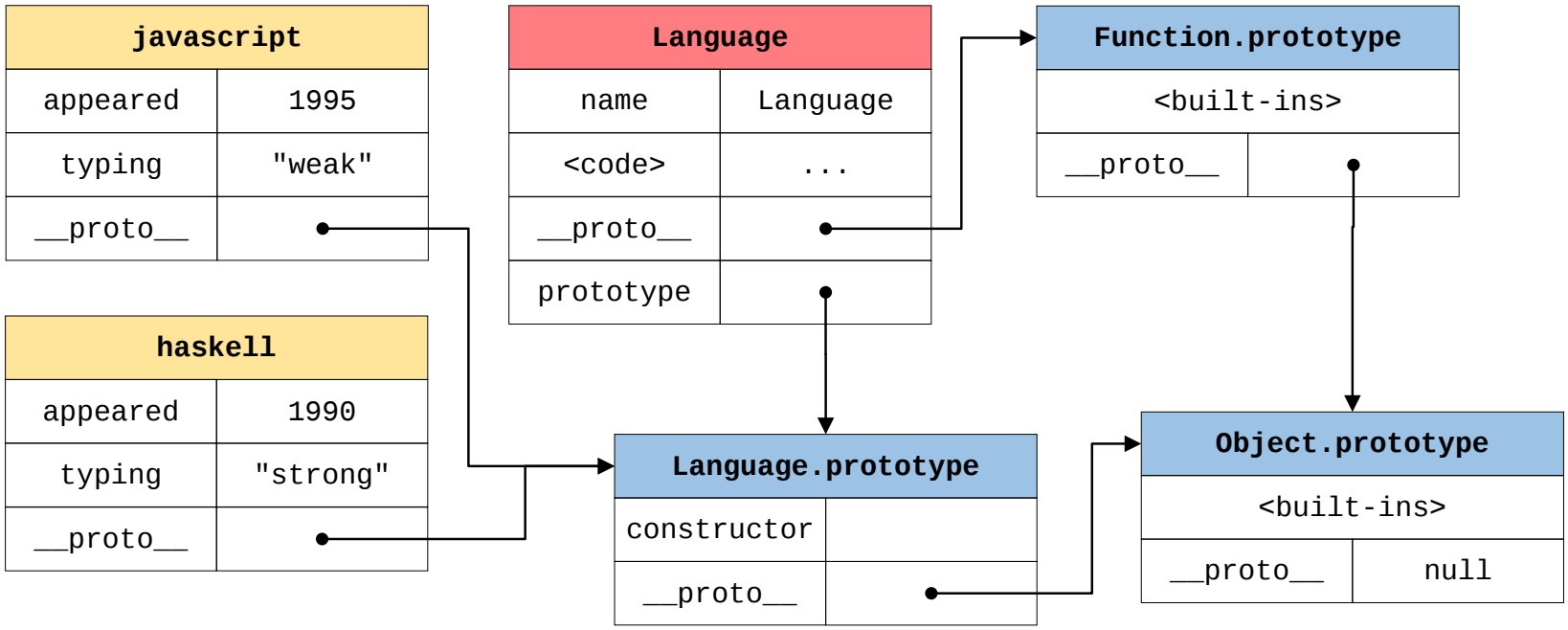
```
javascript = new Language(1995, 'weak')
```



Programmierparadigma

Objektorientierte Programmierung

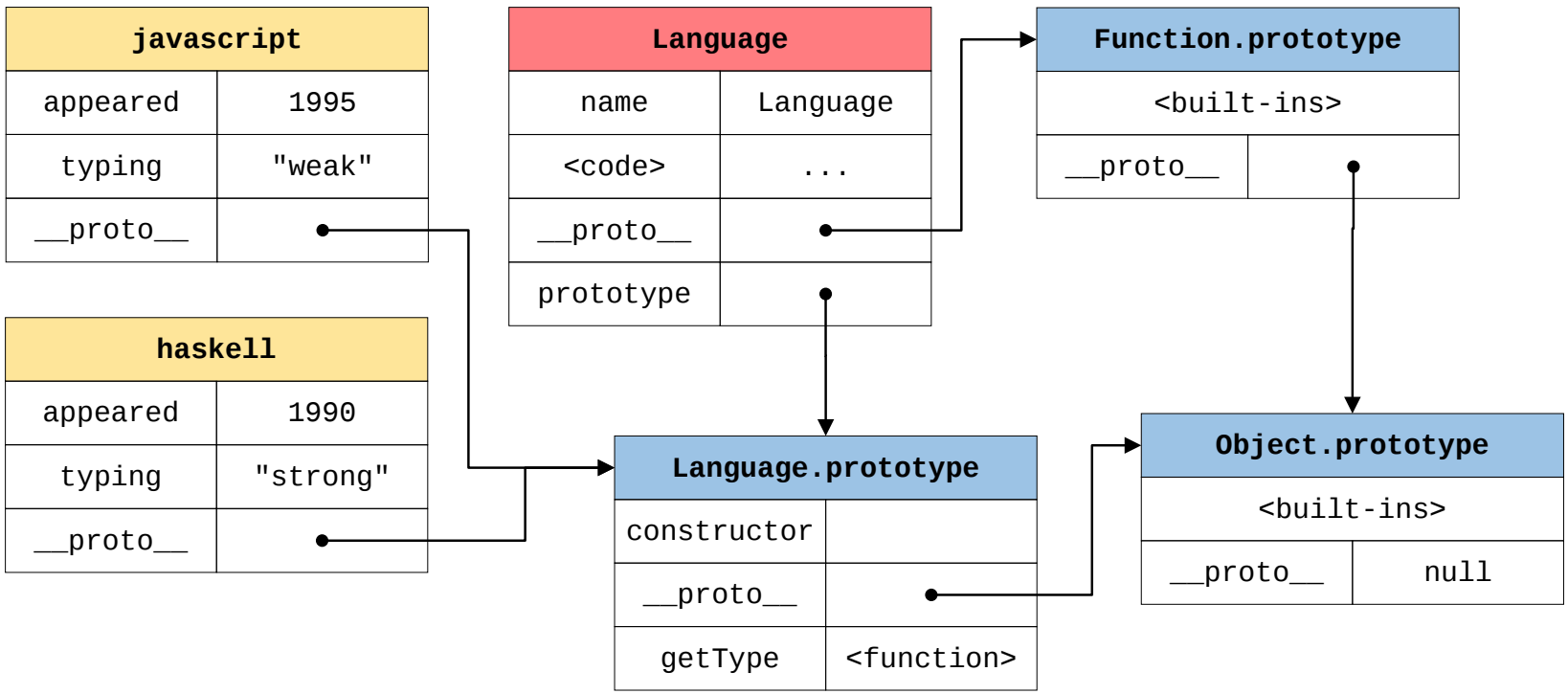
```
javascript = new Language(1995, 'weak')
haskell = new Language(1990, 'strong')
```



Programmierparadigma

Objektorientierte Programmierung

```
Language.prototype.getTyping = function {
  return this.typing
}
```



Programmierparadigma

Funktionale Programmierung

- JavaScript unterstützt funktionale Programmierung
- Keine funktionale Programmiersprache sondern Eigenschaften einer funktionalen Programmiersprache

[6] https://en.wikipedia.org/wiki/Partial_application

Programmierparadigma

Funktionale Programmierung

- JavaScript unterstützt funktionale Programmierung
- Keine funktionale Programmiersprache sondern Eigenschaften einer funktionalen Programmiersprache
  - Unterstützt First-class functions
  - Unterstützt Lambdas
  - Unterstützt *nicht* immutability
  - Unterstützt *keine* algebraische Datentypen
  - Unterstützt *kein* pattern matching
  - Unterstützt *nicht* die partielle Funktionenanwendung[6]

[6] https://en.wikipedia.org/wiki/Partial_application

Programmierparadigma

Funktionale Programmierung

- JavaScript unterstützt funktionale Programmierung
- Keine funktionale Programmiersprache sondern Eigenschaften einer funktionalen Programmiersprache
  - Unterstützt First-class functions
  - Unterstützt Lambdas
  - Unterstützt *nicht* immutability
  - Unterstützt *keine* algebraische Datentypen
  - Unterstützt *kein* pattern matching
  - Unterstützt *nicht* die partielle Funktionenanwendung[6]

[6] `https://en.wikipedia.org/wiki/Partial_application`

Programmierparadigma

Funktionale Programmierung

Eine wesentliche Eigenschaft der funktionalen Programmierung, ist die Komposition von *pure Functions*.

Demo: Funktionale Programmierung mit JavaScript (angelehnt an Haskell). Folgende Einschränkungen werden dazu definiert:

1. Keine Loops
2. Keine `If-else`
3. Eine Funktion besteht aus einem einzigen Return

Programmierparadigma

Funktionale Programmierung

Eine wesentliche Eigenschaft der funktionalen Programmierung, ist die Komposition von *pure Functions*.

Demo: Funktionale Programmierung mit JavaScript (angelehnt an Haskell). Folgende Einschränkungen werden dazu definiert:

1. Keine Loops
2. Keine `If-else`
3. Eine Funktion besteht aus einem einzigen Return
4. Keine Side-Effects
5. Keine Zuweisungen in Funktionen
6. Keine Arrays
7. Funktionen mit 0 oder 1 Argument

Programmierparadigma

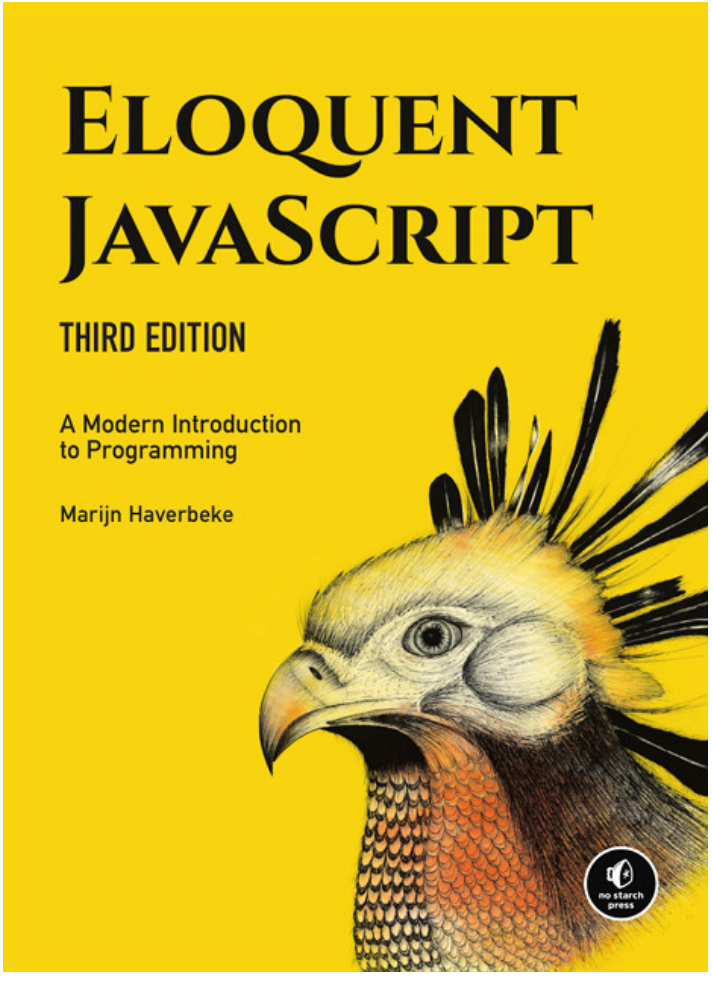
Funktionale Programmierung

Eine wesentliche Eigenschaft der funktionalen Programmierung, ist die Komposition von *pure Functions*.

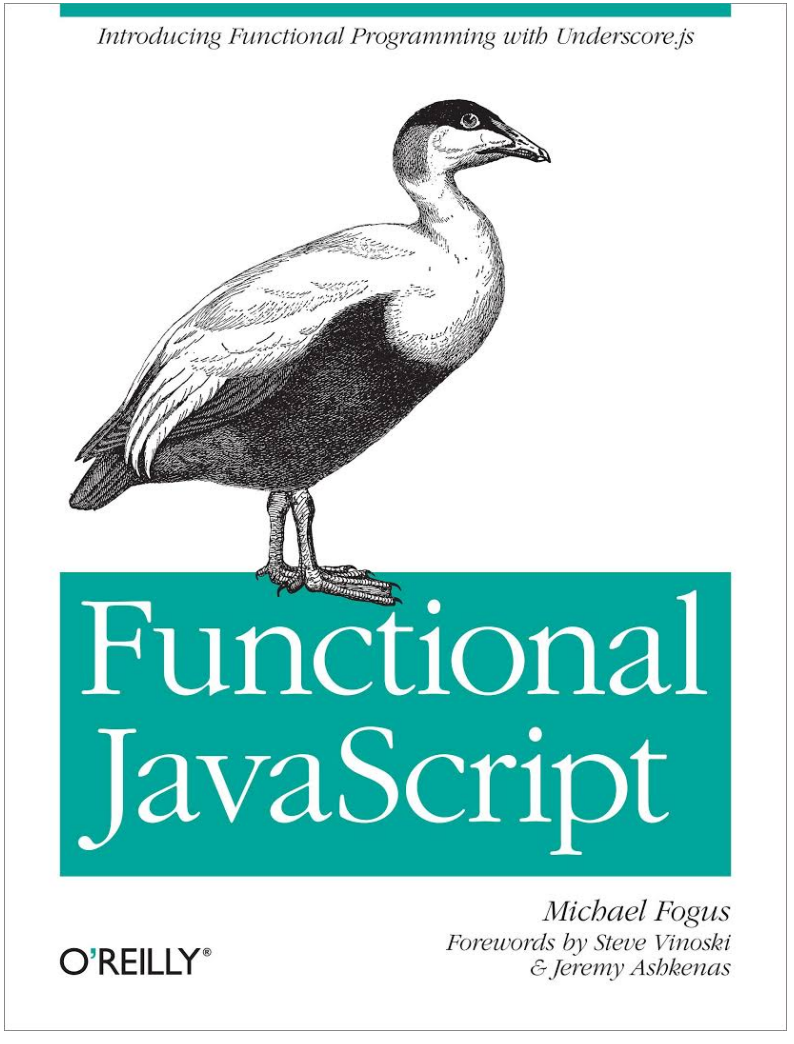
Demo: Funktionale Programmierung mit JavaScript (angelehnt an Haskell). Folgende Einschränkungen werden dazu definiert:

1. Keine Loops
2. Keine `If-else`
3. Eine Funktion besteht aus einem einzigen Return
4. Keine Side-Effects
5. Keine Zuweisungen in Funktionen
6. Keine Arrays
7. Funktionen mit 0 oder 1 Argument

Literatur



`https://eloquentjavascript.net`



Diskussion & Fragen