```
<!Doctype html>
<body>
<h3>js01_var.html</h3>
<br /> Eingabe1:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Eingabe2:<input type="text" id="ip2" name="ip2" value="0"/>
<br /> <input type="button" value="rechnen" onclick=
"//Javascript begin
var ein1=+ip1.value;//
var ein2=+ip2.value;
t1.innerHTML=
'Eingabe1+Eingabe2='+(ein1+ein2)+String.fromCharCode(10)+
'Eingabe1-Eingabe2='+(ein1-ein2)+String.fromCharCode(10)+
'Eingabe1*Eingabe2='+(ein1*ein2)+String.fromCharCode(10)+
'Eingabe1/Eingabe2='+(ein1/ein2)+String.fromCharCode(10)+
'Eingabe1%Eingabe2='+(ein1%ein2)+'(Divisionsrest)'
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="80">
</textarea>
</body>
</html>
<!--
Hier beginnt ein html-Kommentar über mehrere Zeilen
Javascript-Zeilen-Kommentare mit //. (mehrzeilig: /*...*/)
Nach dem Doppelklick erscheinen 2 input-Felder und ein button.
click auf den button startet das Javascriptprogramm, innerhalb
dieses Programms muss man Texte in Hochkommata setzen.
Das Javascriptprogramm wird von HTML (HyperTextMarkupLanguage)
eingerahmt. Der Zusammenhang wird hier über id hergestellt.
Beides wird vom browser (z.B. firefox) interpretiert.
Die Ausgabe erfolgt in ein Textfeld.für sich.
String.fromCharCode(10) ergibt eine neue Zeile.
Bereits hier erkennt man den objektorientierten Ansatz von Javascript.
Man erkennt es an Anweisungen wie String.fromCharCode(10) mit Punkt.
In js11.. werden wir dies ausführlich behandeln, und eigene Objekte
definieren, einstweilen verwenden wir vordefinierte Javascriptobjekte
in der in den Beispielen erklärten Form.
Ausprobieren (firefox):
<html>...</html> markieren Extras Web-Entwickler Javascript-Konsole
einfügen speichern, dann gespeicherte Datei Doppelklicken.
Ausprobieren: Einen Texteditor öffnen und den Text von
<!Doctype html> bis </html> kopieren, unter js.html speichern,
schliessen und doppelklicken. Dann öffnet sich der browser und die
Datei wird ausgeführt.
der hier endet
-->

<!DOCTYPE html>
<body>
<h3>js02_if.html</h3>
<br /> Eingabe1:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Eingabe2:<input type="text" id="ip2" name="ip2" value="0"/>
<br /> <input type="button" value="vergleichen" onclick=
"//Javascript begin
var ein1=+ip1.value;
var ein2=+ip2.value;
if (ein1<ein2){
t1.innerHTML=ein1+'<'+ein2;
}
else if(ein1==ein2){
t1.innerHTML=ein1+'='+ein2;
}
else{
```

```
t1.innerHTML=ein1+'>'+ein2;
}//Javascript end
"
/>
</br><textarea id="t1">
</textarea>
</body>
</html>
<!--
Das Programm erklärt sich selbst, wenn man if=wenn else=sonst übersetzt.
Weitere Vergleichsopperatoren <=, >=, !=(ungleich), (zusätzlich ===)
Der Programmablauf verzweigt sich gemäß Bedigung
ja<Bedingung>nein
>ja-Anweisungen
>nein-Anweisungen
>Programmforsetzung nach Ausführung
entweder der ja- oder nein-Anweisungen.
-->

<!Doctype html>
<body>
<h3>js03_for.html</h3>
<br /> Start:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Ende :<input type="text" id="ip2" name="ip2" value="0"/>
<br /> Step :<input type="text" id="ip3" name="ip3" value="0"/>
<input type="button" value="click" onclick="
var ein1=+ip1.value;//Javascript begin
var ein2=+ip2.value;
var ein3=+ip3.value;
for (i = ein1; i <=ein2; i=i+ein3) {
alert('i='+i);
}
//Javascript end
">
</body>
</html>
<!--
Hier wurde als Ausgabe die alert-Funktion gewählt,
die man bequem abbrechen kann, es kann nämlich Probleme geben.
Hierzu gibt es die prompt-Eingabefunktion (e=prompt("Eingabe");)
Hier und in den nächsten beiden Beispielen wird die
Schleifenstruktur in Programmen behandelt.
Es wird jeweils ein Programmteil mehrfach wiederholt, dies
erfordert eine Abruchbedingung, sonst geht es weiter bis
kein Speicher mehr frei ist.
Bei for und while steht sie am Anfang, bei do-while am Ende.
-->

<!Doctype html>
<body>
<h3>js04_while.html</h3>
<br /> Start:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Ende :<input type="text" id="ip2" name="ip2" value="0"/>
<br /> Step :<input type="text" id="ip3" name="ip3" value="0"/>
<input type="button" value="click" onclick="
var ein1=+ip1.value;//Javascript begin
var ein2=+ip2.value;
var ein3=+ip3.value;
var i=ein1;
while (i<=ein2)
{
alert('i='+i);
i=i+ein3;
}
```

```
//Javascript end
">
</body>
</html>
<!--
Hier wurde als Ausgabe die alert-Funktion gewählt,
die man bequem abbrechen kann, es kann nämlich Abbruch-Probleme geben.
Passend zu alert gibt es die prompt-Eingabefunktion (e=prompt("Eingabe");)
-->

<!Doctype html>
<body>
<h3>js05_do.html</h3>
<br /> Start:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Ende :<input type="text" id="ip2" name="ip2" value="0"/>
<br /> Step :<input type="text" id="ip3" name="ip3" value="0"/>
<input type="button" value="click" onclick="
var ein1=+ip1.value;//Javascript begin
var ein2=+ip2.value;
var ein3=+ip3.value;
var i=ein1;
do{
alert('i='+i);
i=i+ein3;
}while (i<=ein2);
//Javascript end
">
</body>
</html>
<!--
Hier wurde als Ausgabe die alert-Funktion gewählt,
die man bequem abbrechen kann, es kann nämlich Probleme geben.
Hierzu gibt es die prompt-Eingabefunktion (e=prompt("Eingabe");)
-->

<!DOCTYPE html>
<body>
<h3>js06_and.html</h3>
<br /> Eingabe1:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Eingabe2:<input type="text" id="ip2" name="ip2" value="0"/>
<br /> <input type="button" value="positiv?" onclick=
"//Javascript begin
var ein1=+ip1.value;
var ein2=+ip2.value;
if (ein1>0 && ein2>0){
t1.innerHTML=ein1+' und '+ein2+' positiv';
}
else if(ein1>0 || ein2>0){
t1.innerHTML=ein1+' oder '+ein2+' positiv';
}
else{
t1.innerHTML='weder '+ ein1+' noch '+ein2 +' positiv';
}//Javascript end
"
/>
</br><textarea id="t1">
</textarea>
</body>
</html>
<!--
&&: und, ||: oder, !; nicht
-->

<!DOCTYPE html>
```

```
<body>
<h3>js07_case.html</h3>
<br /> Wahl:<input type="text" id="ip1" name="ip1" value="0/1/2"/>
<br /> Eingabe2:<input type="text" id="ip2" name="ip2" value="0"/>
<br /> <input type="button" value="Wahl?" onclick=
"//Javascript begin
var ein2=ip2.value;//jetzt ohne +, dann Text
switch (ein2)
{
  case '0': t1.innerHTML='Wahl='+ein2;
            break;
  case '1': t1.innerHTML='Wahl='+ein2;
            break;
  case '2': t1.innerHTML='Wahl='+ein2;
            break;
  default:  t1.innerHTML='kein Treffer';
}
//Javascript end
"
/>
<textarea id="t1">
</textarea>
</body>
</html>
<!--
Das Programm erklärt sich selbst, wenn man switch=Schalter case=Fall
break=Abbruch default=Voreinstellung übersetzt.(Fallunterscheidung)
Anwendung: Menü
-->

<!DOCTYPE html>
<head>
<script>
function hallo() {
t1.innerHTML="Die Funktion hallo() grüßt!"+String.fromCharCode(10);
}
</script>
</head>
<body>
<h3>js08_function1.html</h3>
<br /> <input type="button" value="Grüß mich" onclick=
"//Javascript begin
hallo();
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="60">
</textarea>
</body>
</html>
<!--
Wir haben hier das einfachste Beispiel einer Javascript-Funktion.
Wie wir noch sehen werden spielen Funktionen eine zentrale Rolle.
Funktionen (auch mehrere) schreibt man normalerweise zwischen
<head><script> und </head></script>
Drückt man den Knopf, wird sie ausgeführt.
-->

<!DOCTYPE html>
<head>
<script>
function hallo() {
return 'hallo';
}
```

```
</script>
</head>
<body>
<h3>js08_function2.html</h3>
<br /> <input type="button" value="Grüß mich" onclick=
"//Javascript begin
t1.innerHTML='Die Funktion hallo() gibt zurück: '+hallo()
+String.fromCharCode(10)
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="60">
</textarea>
</body>
</html>
<!--
Wir haben hier ein einfaches Beispiel einer Funktion mit Rückgabewert.
-->

<!DOCTYPE html>
<head>
<script>
function umfang(param1,param2){
return 2*(param1+param2);
}
function inhalt(param1,param2){
return param1*param2;
}
</script>
</head>
<body>
<h3>js08_function3.html</h3>
<br /> Länge:<input type="text" id="ip1" name="ip1" value="0"/>
<br /> Breite:<input type="text" id="ip2" name="ip2" value="0"/>
<br /> <input type="button" value="berechne Rechteck" onclick=
"//Javascript begin
var ein1=+ip1.value;//
var ein2=+ip2.value;
t1.innerHTML=
'Umfang='+umfang(ein1,ein2)+String.fromCharCode(10)+
'Inhalt='+inhalt(ein1,ein2)+String.fromCharCode(10)
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="60">
</textarea>
</body>
</html>
<!--
Durch umfang(ein1,ein2) wird param1=ein1 und param2=ein2 übergeben.
Zurückgegeben wird dann der Wert von 2*(param1+param2)
Entsprechend Inhalt.
Auf diese Weise kann man sich eine eigene Funktionsbibliothek anlegen.
Diese kann man dann sogar in eine eigene js-Datei auslagern:
<script src="https://dl.dropboxusercontent.com/u/11312988/js_bibliothek.js">
</script>
Der Ort der Bibliothek muss natürlich genau angegeben werden, hier liegt
sie sogar auf Dropbox, und zwar im öffentlichen Ordner meines accounts.
Die beiden Funktionen liegen aber nicht dort.
-->

<!DOCTYPE html>
<head>
<script src="https://dl.dropboxusercontent.com/u/11312988/js_bibliothek.js">
```

```
</script>
</head>
<body>
<h3>js08_function4.html</h3>
<input id="ip1" type="number" min="10" max="200">Gewicht in kg</br>
<input id="ip2" type="number" min="100" max="250">Groesse in cm</br>
<input type="button" value="enter" onclick="
var gw=ip1.value;//begin
var gr=ip2.value;
var bm=bodyMass(gw,gr);//Funktionsaufruf
var kommentar='normal';
if(bm<20)kommentar='zu klein';
if(bm>30)kommentar='zu gross';
t1.innerHTML =bm+', '+kommentar;//end
"/>
</br><textarea id="t1" style="background-color:lightgrey; color:red">
</textarea>
</body>
</html>
<!--
Man beachte hier die Auslagerung der Funktion bodyMass(..,..); man muss aber
die Schreibweise beachten (klein anfangen, dann jedes Wort gross beginnen)
und Parameterreihenfolge wie in der Bibliothek verwenden.
Des weiteren kann man bei dieser input-Form einige Vorgaben machen.
Bei eigener Auslagerung muss man natürlich auch den Ort angeben, wenn
sie nicht im gleichen Ordner liegt wie die html-Datei.
Zusätzlich wurde noch eine css-Anweisung verwendet.
-->

<!DOCTYPE html>
<head>
<script src="https://dl.dropboxusercontent.com/u/11312988/js_bibliothek.js">
</script>
</head>
<body>
<h3>js08_function5.html</h3>
<input id="ip1" type="number" min="10" max="200">Gewicht in kg</br>
<input id="ip2" type="number" min="100" max="250">Groesse in cm</br>
<input type="button" value="enter" onclick="
var gw=ip1.value;//begin
var gr=ip2.value;
var bm=bodyMass(gw,gr);//Funktionsaufruf
var kommentar='normal';
t1.style.color='red';
if(bm<20)kommentar='zu klein';
if(bm>30)kommentar='zu gross';
if(kommentar=='normal')t1.style.color='green';
t1.innerHTML =bm+', '+kommentar;//end
"/>
</br><textarea id="t1" style="background-color:lightgrey;">
</textarea>
</body>
</html>
<!--
Zusätzlich wurden hier noch css-Anweisungen verwendet, zuerst im
Javascriptbereich
dann im html-Bereich.
-->

<!DOCTYPE html>
<body>
<h3>js09_arrays1.html</h3>
<br /> <input type="button" value="Obstangebot" onclick=
"//Javascript begin
```

```
var obst=['Äpfel','Birnen','Citronen','Datteln','Erdbeeren'];
//oder a = new Array();
//a[0]='Äpfel'; ...
var ausgabe='';
for(i=0;i<5;i++){
ausgabe=ausgabe+obst[i]+String.fromCharCode(10);
}
t1.innerHTML=ausgabe
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="20">
</textarea>
</body>
</html>
<!--
Eindimensionale Arrays sind Listen, zweidimensionale Tabellen
->

<!DOCTYPE html>
<body>
<h3>js09_arrays1.html</h3>
<br /> <input type="button" value="Obstangebot" onclick=
"//Javascript begin
var obst=[['Äpfel','Birnen','Citronen','Datteln','Erdbeeren'],
['2.00','2.50','5.00','6.00','3.00']];
//oder a = new Array();
//a[0][0]='Äpfel'; ...
var ausgabe='';
for(i=0;i<5;i++){
ausgabe=ausgabe+obst[0][i]+':'+obst[1][i]+'/kg'+String.fromCharCode(10);
}
t1.innerHTML=ausgabe
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="20">
</textarea>
</body>
</html>
<!--
Eindimensionale Arrays sind Listen, zweidimensionale Tabellen
Die Tabelle (Matrix) hätte folgendes Aussehen:
['Äpfel','Birnen','Citronen','Datteln','Erdbeeren']Zeile 0
['2.00' ,'2.50'  ,'5.00'    ,'6.00'    ,'3.00'    ]Zeile 1
Spalte 0  Spalte 1 Spalte 2   Spalte 3  Spalte 4
->

<!DOCTYPE html>
<body>
<h3>js10_string.html</h3>
<br /> <input type="button" value="aBcDeFgHiJkL" onclick=
"//Javascript begin
var s='aBcDeFgHiJkL';
var ausgabe='';
ausgabe=ausgabe+'s='+s+String.fromCharCode(10)+
's.length='+s.length+String.fromCharCode(10)+
's.substr(4,5)='+s.substr(4,5)+String.fromCharCode(10)+
's.substring(1,4)='+s.substring(1,4)+String.fromCharCode(10)+
's.charAt(0)='+s.charAt(0)+String.fromCharCode(10)+
's.toUpperCase='+s.toUpperCase()+String.fromCharCode(10)+
's.toLowerCase='+s.toLowerCase()+String.fromCharCode(10)+
's='+s+' unverändert!';
t1.innerHTML=ausgabe
```

```
;//Javascript end
"
/>
</br><textarea id="t1" rows="10" cols="40">
</textarea>
</body>
</html>
<!--
Strings sind Zeichenketten bzw. Texte, man muss sie in
Anführungszeichen schreiben.
->

<!DOCTYPE html>
<head>
<script>
function Person () {
  this.vname;
  this.name;
  this.name=function(){
  return this.vname+":"+this.nname
};
}
</script>
</head>
<h3>js11_obj1.html</h3>
<body>
<p id="p1">
</p>
<script>
var pers1= new Person();
pers1.vname="Udo";
pers1.nname="Kaden";
p1.innerHTML=
'pers1.vname='+pers1.vname+'</br>'+
'pers1.nname='+pers1.nname+'</br>'+
'pers1.name='+pers1.name();
</script>
</body>
</html>
<!--
Zwischen <head><script>und</scrip></script> steht hier
der Konstruktor des Prototyps des Objekts Person.
Das Hauptprogramm steht hier im html-body, ebenfalls zwischen
<script> und </script>.
In Java würde man dies den Konstruktor der Klasse Person nennen.
Zudem wäre die Klasse abstrakt.
Die Variablen eines Prototyps heissen properties (Eigenschaften),
die Funktionen heissen methods (Methoden).
Javascript ist eine objektorientierte Programmiersprache.
Wir haben bereits in den meisten früheren Beispielen objektorientiert
programmiert, nämlich immer dann wenn wir die Punktverknüpfung
benutzt haben, z.B. s.length ergab die Länge des Stings s.
Strings sind ebeso wie Arrays in Javascript Objekte.
Der Clou ist, dass man html-Elemente über ids als Objekte ändern
kann. Das haben wir von Anfang an gemacht, insbesondere bei der Ausgabe.
-->

<!DOCTYPE html>
<head>
<script>
function Person () {
  this.vname;
  this.name;
  this.name=function(){
```

```
  return this.vname+":"+this.nname
};
}
</script>
</head>
<body>
<h3>js11_obj2.html</h3>
<p id="p1">
</p>
<script>
var ausgabe="";
var pa = new Array();
pa[0]=new Person();
pa[0].vname="Udo";
pa[0].nname="Kaden";
pa[1]=new Person();
pa[1].vname="Ilona";
pa[1].nname="Kaden";
for(var i=0;i<2;i++){
ausgabe=ausgabe+"pa["+i+"].name()="+pa[i].name()+"</br>";
}
p1.innerHTML=ausgabe;
</script>
</body>
</html>
<!--
Das p-Element muss hier vor dem html-Teil stehen.
Das Problem bei diesem Programm ist, dass die Daten im
Programm stehen. Mit einfachem Javascript ist die Datenspeicherung
nur auf dem eigenen Computer mittels localstorage möglich, aber nur
in der Form key/data, was aber programmtechnisch beherrschbar ist.
Jedoch gibt es nur eine Datenbank, die zudem durch ein Programm
wie ccleaner gelöscht wird.
-->

<!DOCTYPE html>
<head>
<script>
function Person () {//Elter
  this.vname;
  this.nname;
  this.name= function(){//this.name=this.vname+this.nname ergibt ""
return this.vname+":"+this.nname
};
}//Elter
function Manager () {//Kind1
  this.berichte = [];
}
Manager.prototype = new Person;//Kind1
function Arbeiter () {//Kind2
  this.projekte = [];
}
Arbeiter.prototype = new Person;//Kind2
</script>
</head>
<body>
<h3>js11_obj3.html</h3>
<p id="p1">
</p>
<script>
var m1=new Manager();
m1.vname="Udo";
m1.nname="Kaden";
m1.berichte=["b1","b2"];
```

```
var a1=new Arbeiter();
a1.vname="Ilona";
a1.nname="Kaden";
a1.projekte=["p1","p2"];
p1.innerHTML=m1.name()+" "+m1.berichte[0]+"</br>"+
a1.name()+" "+a1.projekte[1];
</script>
</body>
</html>
<!--
Zunächst wird wieder der Prototyp Person wie in js11_obj1.html angelegt.
Die anschließend angelegten Prototype Manager und Arbeiter
werden als Erweiterung von Person angelegt. Sie haben jeweils
eine zusätzliche Methode. Daneben können sie aber auch über
die Eigenschaften von Person verfügen. Diese könnten auch neu
definiert werden. Ebenso könnten ausgehen von Manager und Arbeiter
weitere Erweiterungen vorgenommen werden. Den Vorgang nennt man
fälschlicherweise Vererben, da ja von Person nach wie vor Prototype
gebidet werden können.
In JAVA: class Manager extends Person{...}
-->

<!DOCTYPE html>
<head>
<script>
function Person () {//Elter
  this.vname;
  this.nname;
  this.name= function(){//this.name=this.vname+this.nname ergibt ""
return this.vname+":"+this.nname
};
}//Elter
function Manager () {//Kind1
  this.berichte = [];
}
Manager.prototype = new Person;//Kind1
function Arbeiter () {//Kind2
  this.projekte = [];
}
Arbeiter.prototype = new Person;//Kind2
</script>
</head>
<body>
<h3>js11_obj4.html</h3>
<p id="p1">
</p>
<script>
var m1=new Manager();
m1.vname="Udo";
m1.nname="Kaden";
m1.berichte=["b1","b2"];
var a1=new Arbeiter();
a1.vname="Ilona";
a1.nname="Kaden";
a1.projekte=["p1","p2"];
p1.innerHTML=m1.name()+" "+m1.berichte[0]+"</br>"+
a1.name()+" "+a1.projekte[1];
</script>
</body>
</html>
<!--
Zunächst wird wieder der Prototyp Person wie in js11_obj1.html angelegt.
Die anschließend angelegten Prototype Manager und Arbeiter
werden als Erweiterung von Person angelegt. Sie haben jeweils
```

```
eine zusätzliche Methode. Daneben können sie aber auch über
die Eigenschaften von Person verfügen. Diese könnten auch neu
definiert werden. Ebenso könnten ausgehen von Manager und Arbeiter
weitere Erweiterungen vorgenommen werden. Den Vorgang nennt man
fälschlicherweise Vererben, da ja von Person nach wie vor Prototype
gebidet werden können.
In JAVA: class Manager extends Person{...}
-->

<!DOCTYPE html>
<head>
<script>
function UDF() {
this.kreisInhalt = function(r) {
return (Math.PI*r*r);
};
this.kreisUmfang = function(r) {
return (Math.PI*r*2);
};
}
var meinUDF = new UDF();
</script>
</head>
<body>
<h3>js11_obj5_udf.html</h3>
Radius:<input id="ip1" type="text">
<input type="button" value="Kreis_berechnen" onclick="
var r=ip1.value;
p1.innerHTML =
'Kreisinhalt ist ' + meinUDF.kreisInhalt(r)+'</br>'+
'Kreisumfang ist ' + meinUDF.kreisUmfang(r);
"/>
<p id="p1"></p>
</body>
</html>
<!--
Dieses Beispiel zeigt nochmals die Auslagerung von Funktionen.
Mittels Vererbung könnte man diese übersichtlich strukturieren.
-->
```