JavaScript

In dieser Scriptsprache werden Anweisungen direkt , Schritt für Schritt, ausgeführt. Hingegen wird bei C oder Java der Code zuerst compiliert.

1 Zeichen, Kommentare und Begriffe

- Steuerzeichen

\n	NewLine – Cursor wird zum Anfang der nächsten Zeile bewegt
\r	Carriage Return – Cursor geht zum Anfang der aktuellen Zeile
\t	Hotizontal Tab – Cursor geht zur nächsten horizontalen Tab-Position
\v	Vertival Tab – Cursor geht zur nächsten vertikalen Tab-Position
\o	Endmarkierung eines Strings (letzte Zeichen)
\“	Zeichen „ wird ausgegeben (weitere Ausgaben - \´; \?; \\)
\xhh	Hexadezimalwert wird ausgegeben (hh ist der Hex-Wert)
\nnn	Oktalwert wird ausgegeben (nnn ist der Oktalwert)

- Kommentare

```
//      eine ganze Kommentarzeile
Var a  // ab hier beginnt der Kommenar
/*      erste Zeile
 *      Weitere Zeile vom Kommentar
*/      (Ende des Kommentarbereiches)
```

- Begriffe

Bezeichner	Namen für Objekte in Programmen (Variablen, Funktionen)
Schlüsselwörter	reservierte Wörter (while, if)
Begrenzer	Symbole die Programmstrtukturen abgrenzen
Semikolon	schliet eine Anweisung ab
Komma	trennt Argumente einer Parameterliste
()	begrenzt Ausdrücke in Kontrollstrukturen
{}	umschließt Anweisungsblöcke
[]	Tabellenelemente
=	weist einer Variablen einen Wert zu (a=1)
===	Vergleich von Werten (if a === 9)

2 Daten und Datentypen

number	eine ganze Zahl oder Fließkommazahl (8 oder 2.34)
string	eine Squenz von Zeichen, die einen Text darstellen
boolean	Wahrheitswert (true oder false)
object	ein Name-Wert-Paar; ein Datum; eine Tabelle
undefined	Variable, dessen Wert nicht definiert ist
null	Schlüsselwort das einen null-Wert kennzeichnet

3 Variablen und Kostanten

- Deklaration

Das ist der Vorgang um eine Variable bekannt zu machen. Dazu gibt es die Schlüsselwörter *var* und *let*, die aber nicht unbedingt benutzt werden müssen. (x; var x; let x) Der Unterschied besteht in der Sichtbarkeit.

- Definition

Dies ist der Vorgang um einer Variablen Speicherplatz im Programm zuzuweisen und mit einem Wert zu belegen.

let x;	undefiniert
str = ‚Hallo‘;	Initialisierung String
Seuer =‘ \n‘;	Initialisierung mit Zeilenvorschub
num = 9;	Initialisierung mit 9
bool = true;	Initialisierung mit wahr
obj = {„Name“ : „wert“};	Initiaisierung mit Objekt

- Gültigkeisbereich von Variablen

Damit wird beschrieben, wo eine Variable innerhalb des Programms nutzbar und sichtbar ist. Es gibt globale und lokale Variablen. Context – nur in der entsprechenden Node verfügbar; flow – im gesamten flow verfügbar; global – Flow übergreifend verfügbar

- Konstanten

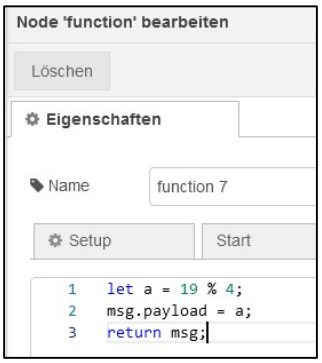
Sie beginnen mit dem Schlüsselwort *const*. Sie benötigt bei der Deklaration unbedingt einen Wert.
const PI = 3.14, UD = 0.91 (z.B. Umrechnungsfaktor Dollear Euro)

4 Ausdrücke und Operationen

Ein Ausdruck ist eine Verknüpfung aus Operanden (Variablen oder Rückgabewerte von Funktionen). Daraus erbibt sich dann ein neuer Wert. (x = y * z). Operationen können unär oder binär sein.
unär nur ein Operant x = -x (das Ergebnis wird negiert)
binär zwei Operanten x = y * z
Ausdrücke können in Klammern geschachtelt werde. Dabei gilt: Punkt vor Strichrechnung. x = (a - b) * (h + e);

- Arithmetische Operationen

let i = 3 + 4; Addition, Subtraktion, Division, Multiplikation (7)
let i = 19 % 4; Division zweier Ganzzahlen mit Ausgabe des Restes (Modulu) (3)
let i = 4 * 4; Potenzfunktion (16)
let i = 4 ** (1/2); Quadratwurzel (2)



Zum Testen wird die funktion-Node verwendet

- Operationen

let w = 7.88; einfache Zuweisung (7.88)
let h = „A“; einfache Zuweisung („A“)
let m = „Hallo“; einfache Zuweisung („Hallo“)
i++; Inkement anstelle von i = i +1, wenn i=1 -> (2)
r--; Dekrement anstelle von r = r -1, wenn r=4 -> (3)
let s = „O“ + „K“; Sring zusammensetzen mit +, („OK“)
let m = (7 == 2); bedingte Operation – Gleich u. Ungleich, Ergebnis: m = false (false)
let n = (5 == 5); bedingte Operation - Gleich u. Ungleich, Ergebnis: n = true (true)
let v = (7<2); bedingte Operation – Größer und Kleiner, Ergebnis: v = false (false)
let w = (7>2); bedingte Operation – Größer und Kleiner, Ergebnis: w = true (true)
let x = (4<5) && (2<3) Logisches UND (&&), Ergebnis des Ausdruckes ist 1, wenn alle Teilergebnisse 1 sind. (true)
let x =(4<5) || (2<3) Logisches ODER (||), Ergenis des Ausdruckes ist 1, wenn ein Teilergebnisse 1 ist. (true)
let z = (f==9) ? true : false; Bedingunsoperator. Es wird eine Bedingung ausgewertet. Ist die Bedingung wahr gilt der erste Ausdruck, ansonsten der Zweite. f=9 (true); f=8 (false)

- Typenumwandlung

let t = String(78); Zahl in String („78“)
let t = (78 + 1).toString(); Zahl in String („79“)
let t = (7.8927).toFixed(2); Zahl in String mit Begrenzung nach dem Komma („7.89“)
let x = String(false); Boolean in String (“false”)
let z = String(Date()); Datumsangabe in Sring (“Sat Sep 14 2024 08:44:28 GMT+0200 (MEZ)“)
let a = Number(„81“); String in Zahl (81)
let a = parseFloat(“78.4”); String in Zahl (78.4)
let a = parseInt(78.4); String in Zahl mit Ausgabe als Ganzzahl (78)
let s = „47“ + 11; automatische Wandlung, ist ein String enthalten, ist das Ergebnis ein String („4711“)
let s = 4 + 5 + „7“; automatische Wandlung bei komplexer Operation, String hinten („97“)
let s = „6“ + 3 + 2; automatische Wandlung bei komplexer Operation, String vorne („622“)

5 Array-Objekt

Tabellen oder Arrays sind Strukturen von aufeinanderfolgenden Werten. Der Zugriff erfolgt über einen Index.

var Bier = [„Bock“, „Pils“, „Hell”];	Erstellung mit Literal ([„Bock“, „Pils“, „Hell“])
var Bier = new Array („Bock, “Pils”, “Hell”);	Erstellung mit Array-Funktion ([„Bock“, „Pils“, „Hell“])
len n = Bier.length;	Ausgabe der Länge des Arrays (3)
var Sorte = Bier[2];	Auf Element zugreifen. Das erste Element hat den Index 0 („Hell“)
Bier.unshift(„Weizen“);	Element am Anfang hinzufügen ([„Weizen“, „Bock“, „Pils“, „Hell“])
Bier.push(„Koelsch“);	Element am Ende hinzufügen ([„Bock“, „Pils“, „Hell“, „Koelsch“])
Bier.shift();	Element am Anfang löschen ([„Pils“, „Hell“])
Bier.pop();	Element am Ende löschen ([„Bock“, „Pils“])
let pos = Bier.indexOf(„Pils”);	Index eines Elementes ermitteln (1)
Bier.splice(pos, 1);	Angegebenes Element löschen ([„Bock“, „Hell“])
let kopieBier = Bier.slice();	Kopie v. Arrays anlegen, Ausgabe kopieBier ([„Bock“, „Pils“, „Hell“])
let sortBier = Bier.sort();	Inhalt alphabetisch sortieren, sortBier ([„Bock“, „Hell“, „Pils“])

6 Date-Objekt

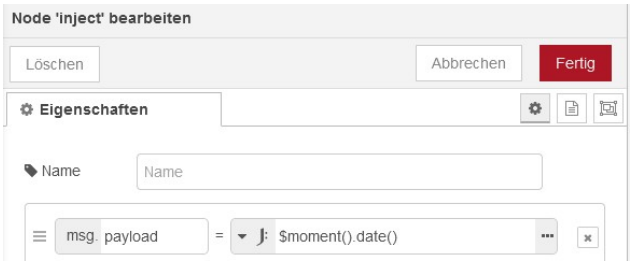
Dieses Objekt ist für alle Berechnungen mit Datum und Uhrzeit zuständig.

let zeit = new Date();	Zeitobjekt ("Sat Sep 14 2024 17:14:43 GMT+0200 (Mitteleuropäische Sommerzeit)")
let m = zeit.getDate();	Datumsangabe aus dem Zeitopbjekt ermitteln, Monatstag (14)
let d = zeit.getDay();	Datumsangabe aus dem Zeitopbjekt ermitteln, Wochentag (6)
let y = zeit.getFullYear();	Datumsangabe aus dem Zeitopbjekt ermitteln, Wochentag (2024)
let t = zeit.getTime();	Zeit seit dem 01.01.1970 in Sekunden (1726327575335)
let h = zeit.getHours();	Stundenteil (18)
let m = zeit.getMilliseconds();	Millisekunden(170)
let o = zeit.getMonth();	Monat (8) Die Monate zählen von 0-11
let u = zeit.getMinutes();	Minutenteil (23)
let s = zeit.getSeconds();	Sekundenteil (34)
let d = Date.now();	Zeit seit dem 01.01.1970 in Sekunden (1726327739289)

Hinweis:

Datumsangaben können auch aus der inject-Node übergeben werden

Millisekunden:	$moment().millisecond()
Sekunden:	$moment().second()
Minuten:	$moment().minute()
Stunde:	$moment().hour()
Monatsdatum:	$moment().date()
Monat als Nummere:	$moment().month() + 1
Jahr:	$moment().year()
Wochentag als Nr.:	$moment().isoWeekday()



7 Funktionen

Funktionen sind Programmierelemente, dessen Merkmal es ist, das sie Teilaufgaben des Programms ausführen. Funktionen sind damit Blöcke von Anweisungen mit einem Namen, der im Programm mehrfach aufrufbar ist.

function addition (a,b)	Funktionskopf mit Funktionsname und Übergabeparameter
{ return a+b}	eigentliche Funktion mit return
msg.payload = addition(2,3);	Aufruf der Funktion mit 2 Übergbewerten für payload
msg.topic = addition(4,8);	Aufruf der Funktion mit 2 Übergbewerten für topic
return msg;	Ausgabe (payload: 5 topic: 12)
	altenative Möglichket
function addition (a,b)	Funktionskopf mit Funktionsname und Übergabeparameter
{ return a+b}	eigentliche Funktion mit return
node.send({"payload": addition(5,1)});	Ausgabe (6)

8 Kontrollstrukturen

Kontrollstrukturen bestimmen die Reihenfolge der abzuarbeitenden Befehle.

- If-Bedingung

Hier wird der Inhalt einer Variablen verglichen und je nach Inhalt eine Aktion ausgeführt.

```
var a = 9;                          Variable wird angelegt
if(a == 9)                          die eigentliche if-Bedingung
{var erg = "a ist gleich 9"}        diese Anweisung wird ausgeführt, wenn if-bedingung wahr ist
msg.payload = erg;                  Inhalt wird dem payload übergeben;
return msg;                         Ausgabe: wahr -> „a ist gleich 9“; falsch -> undefined
```

- if … else

Dies ist eine Erweiterung der einfachen if-Anweisung. Ist das if-Ergebnis false, wird der zweite Codeblock ausgefürt.

```
var a = 9;                          Variable wird angelegt
if(a == 9)                          die eigentliche if-Bedingung
{var erg = "a ist gleich 9"}        diese Anweisung wird ausgeführt, wenn if-bedingung wahr ist
else                                else-Zweig
{ var erg = "a ist nicht gleich 9" }diese Anweisung wird ausgeführt, wenn if-bedingung falsch ist
msg.payload = erg;                  Inhalt wird dem payload übergeben;
return msg;                         Ausgabe: wahr -> „a ist gleich 9“; falsch -> „a ist nicht gleich 9“
```

- switch … case

Mit switch wird eine Fallunterscheidung eingeleitet. Es wird ein zu prüfender Wert vorgegeben der in einer Liste von Vergleichswerten untersucht wird. Ist der Vergleich true, wird der folgende Eintrag abgearbeitet.

```
let Note = 1;                                   Variable wird angelegt
switch (Note) {                                 Kontrollstruktur (Schalter) Klammer öffnen
case 1: node.send ({„payload“ : „sehr gut“});   Anweisungsblock (Fall)wenn Note gleich 1
break;                                          die Kontrollstruktur wird abgebrochen
case 5: node.send ({„payload“ : „mangelhaft“}); Anweisungsblock (Fall)wenn Note gleich 5
}                                               Klammer zum Schließen der Kontrollstruktur
```

- while-Schleife

Sie wiederholt Anweisungen solang die Bedingung *true* liefert.

```
let i = 0;                                  Variable wird angelegt
while (i<2){                                Kontrolle, ob Schleife fortgeführt wird
  node.send({ "payload": "Wert "+i });      Ausführung wenn while = true
i++}                                        Erhöhung der Variable um 1 ( „Wert 0“ u. „Wert 1“)
```

- do-while-Schleife

Der Unterschied zu while ist, dass der Code in den geschweiften Klammern auf jeden Fall einmal ausgeführt wird.

```
let i = 0;                                  Variable wird angelegt
do {                                        Beginn der Schleife, inder die Ausführung mind. 1 mal erfolgt
   node.send({ "payload": "Wert " + i });   Ausführung
   i++}                                     Erhöhung der Variable um 1
while (i < 2);                              Kontrolle, ob Schleife fortgeführt wird   („Wert 0“ u. „Wert 1“)
```

- for-Schleife

Sie wird verwendet, wenn eine bestimmte Anzahl an Durchläufen benötigt wird.

```
let i;                                      Variable wird initialisiert
for (i=0; i <2; i++) {                      Schleifenkopf mit Variable; Bedingung und Erhöhung von i um 1
   node.send ({"payload": "Wert " + i})     Ausführung der Schleife, wenn Bedinung true („Wert 0“ u. „Wert 1“)
}
```