Binär- und Hexadezimal-Zahl Arithmetik.

Prof. Dr. Dörte Haftendorn, MuPAD 4, http://haftendorn.uni-lueneburg.de Aug.06
Automatische Übersetzung aus MuPAD 3.11, 24.04.02 Version vom 12.10.05
Web: http://haftendorn.uni-lueneburg.de www.mathematik-verstehen.de
++

Binär dargestellte Zahlen, auch **Dualzahlen** genannt, und **Hexadezimalzahlen** kommen im Zusammenhang mit Computern häufig vor.
Daher ist es für den Informatiker wichtig, die Grundlagen zu verstehen.
Man sollte auch verstehen, dass man mit den so dargestellten Zahlen **rechnen** kann.
Arithmetik = Lehre von den Rechenverfahren.
Inhalt dieser Datei: **Zahldarstellung LEVEL 1**
Zahldarstellung LEVEL 2
Rückwärtsumwandlung LEVEL 1
Rückwärtsumwandlung LEVEL 2
Zusammenhang zwischen Dualzahlen und Hexzahlen
Zusammenhang zwischen Hexzahlen und Farben in HTML
u.a. Computeranwendungen
Rechnen mit Dualzahlen LEVEL 1
Rechnen mit Dualzahlen LEVEL 2
Rechnen mit Dualzahlen, Komplementaddition LEVEL 3

Zahldarstellung, LEVEL1
Umwandlung von Ganzen Zahlen (=Integerzahlen) in andere Darstellungen (in MuPAD)

```
int2text(35,2);  //verwandle 35 in eine Dualzahl
```

"100011"

Man sieht, dass das Ergebnis eine Zeichenkette (ein String) ist. Das soll in dieser Lern-Datei als Kennzeichnung von den nicht-dezimal dargestellten Zahlen dienen. Um welche Darstellung es sich handelt soll aus dem Kontext hervorgehen.
Lies den Befehl: "**integer to text**" also "Wandlung Integer zu Text" (2 als "to" wie bei den Rucksäcken 4YOU)

Liste von Quadratzahlen in dezimaler, in dualer und in hexdezimaler Darstellung

```
i^2 $ i=1..15
```

1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225

```
int2text(i^2,2) $ i=1..10
```

"1", "100", "1001", "10000", "11001", "100100", "110001", "1000000", "1010001", "

```
int2text(i^2,16) $ i=1..15
```

"1", "4", "9", "10", "19", "24", "31", "40", "51", "64", "79", "90", "A9", "C4", "

Für die Darstellung im Hex-System werden bekanntlich für die Ziffern 10 bis 15 die Buchstabe A bis F verwendet.

```
int2text(i,16) $ i=1..20
```

```
int2text(i,16) $ i=1..20
```

"1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "10", "

Zahldarstellung LEVEL 2

Mathematisch lässt sich jede Zahl g zur Grundlage einer Zahldarstellung machen, man sagt dann: g-adische Darstellung.
Dabei gibt es g Ziffern. Für g >10 bis g<=36 werden nachfolgend die Buchstaben aus dem Alphabet genommen.
In MuPAD gibt es für die Umwandlung zwei Befehle:

```
int2text(9,5)  //Also: 9 ist (1 mal 5 + 4)
```

"14"

```
numlib::g_adic(9,5) //Also: 9 ist (4 + 1 mal 5)  Anders herum wie oben!
```

[4, 1]

```
int2text(100,36)  //Also: 100  ist (2 mal 36 + (9+19),S ist der 19. Buchstabe)
```

"2S"

```
//int2text(100,37)  //Klappt nicht, weil die Buchstaben nicht reichen.
numlib::g_adic(100,37) //Also: 100 ist 26 + 2 mal 37
```

[26, 2]

Der Befehl aus dem Package "Zahlentheorie" ist also umfassender anwendbar.

Rückwärtsumwandlung LEVEL 1

```
text2int("10110101",2)
```

181

```
1*2^7+0*2^6+1*2^5+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0 //"von Hand"
```

181

```
text2int("b5",16)
```

181

```
11*16+5//"von Hand"
```

181

```
text2int("376B",16)
```

14187

```
text2int("11011101101011",2)
```

14187

```
text2int("110011101",2)
```

413

Rückwärtsumwandlung LEVEL 2

Rückwärtsumwandlung LEVEL 2

Ins und aus dem 60-iger System der Babylonier ins Dezimasystem:

```
babylon:=numlib::g_adic(400000,60)
```

[40, 6, 51, 1]

```
40+6*60+51*60^2 +1*60^3  //von Hand
```

400000

Definition einer Funktion, die eine "babylonische Zahlenliste" in eine Dezimalzahl verwandelt.

```
babylon2dez:=proc(bab)
             begin
                 dez:=0;
                 for i from 1 to nops(bab) do
                    dez:=dez+bab[i]*60^(i-1)
                  end_for;
                 return(dez)
             end_proc:
babylon2dez(babylon)
```

400000

Definition einer Funktion, die eine "g_adische Zahlenliste" in eine Dezimalzahl verwandelt.

```
g_adic2dez:=proc(g_zahl,g)
             begin
                 dez:=0;
                 for i from 1 to nops(g_zahl) do
                    dez:=dez+g_zahl[i]*g^(i-1)
                  end_for;
                 return(dez)
             end_proc:

g_adic2dez(babylon,60)
```

400000

```
ziemlich_gross:=numlib::g_adic(65530,2) //als Dualzahl umgekehrt zu lesen!!!
```

[0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```
g_adic2dez(ziemlich_gross,2)
```

65530

```
2^16 -1//soviele positive ganze Zahlen kann man mit 2 Byte darstellen.
```

65535

```
int2text(65535,2)
```

"1111111111111111"

Funktion, die eine g-adische Zahlenliste umdreht:

```
g_adic2zahl:=proc(g_liste,g)
              local li,la, i;
              begin
                 li:=[];la:=nops(g_liste);
                 for i from 1 to nops(g_liste) do
                    li:=li.[g_liste[la+1-i]]
                 end_for;
```

```
                    end_for;
                    return(li);
                end_proc;
```

proc $g_{adic2zahl(g_{liste, g)}}$ ... end

```
g_adic2zahl(babylon,60)
```

[1, 51, 6, 40]

```
g_adic2zahl(ziemlich_gross,2)
```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0]

```
int2text(65530,2)
```

"1111111111111010"

Zusammenhang zwischen Dualzahlen und Hexzahlen

Da man Dualzahlen wegen ihrer vielen Einsen und Nullen so schecht lesen kann ohne sich zu verzählen,
werden sie der Informatik in 4-Blöcke aufgeteilt. Jeder 4-Block wird dann eine hexadezimale Ziffer umgewandelt.
Das klappt, weil "1111" dual=15 dez =F hex, also weil 2^4 -1=16 -1 ist.

```
[i,int2text(i,2),int2text(i,16)] $ i=1..8      // Dez, dual, hex
```

[1, "1", "1"], [2, "10", "2"], [3, "11", "3"], [4, "100", "4"], [5, "101", "5"], [6, "110

```
[i,int2text(i,2),int2text(i,16)] $ i=9..15
```

[9, "1001", "9"], [10, "1010", "A"], [11, "1011", "B"], [12, "1100", "C"], [13, "1101"

```
text2int("101001011101",2),text2int("A5D",16)
//A5D kann man besser lesen als die Dualzahl
```

2653, 2653

Zusammenhang zwische Hexzahlen und Farben in HTML u.a. Computeranwendungen

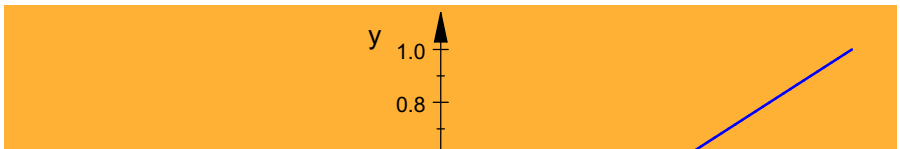
Ein Byte hat 8 Bit, also zwei 4-Blöcke, also zwei Hex-Ziffern, wenn man im RGB-Sytem (Rot-Grün-Blau) für eine Grundfarbe also 1 Byte verwendet, kann man sie Farbe in 2^8 Stärken angeben, also von 0 bis 255 (in vielen Malprogrammen) oder von 00 bis FF Hexdezimal in HTML.
Die Angabe color="#FFAA35" in HTML heißt also:

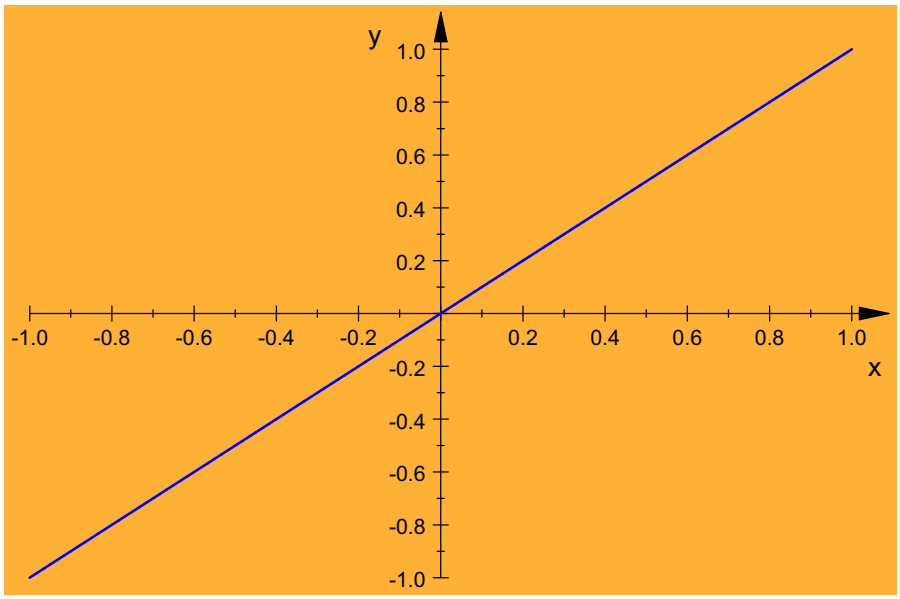
```
text2int("FF",16),text2int("AA",16),text2int("35",16)
```

255, 170, 53

Rot in voller Stärke, reichlich Grün (177/255) und Blau in Stärke 53/255 also etwa 20%
Dieselbe Farbe wird in den meisten CAS mit RGB(1,0,0.2) angeben, also jede Farbe in Prozent der vollen Stärke 1.

```
plot(plot::Function2d(x,x=-1..1,BackgroundColor=[1,177/255,53/255]))
```



Rechnen mit Dualzahlen LEVEL 1

Da in MuPAD die Dualdarstellung mit Strings oder Listen realisiert wird, muss man sich die Rechenoperationen
sebst definieren.

```
plus_bin:=(x,y)->(int2text(text2int(x, 2)+text2int(y,2),2)):
plus_bin("10110","101")
```

"11011"

```
mal_bin:=(x,y)->(int2text(text2int(x, 2)*text2int(y,2),2)):
mal_bin("10110","101")
```

"1101110"

```
int2text(-10,2)    // Das Negative einer Dualzahl
```

"–1010"

```
plus_bin("10110","-101")  //Da ist dann die Subtraktion
```

"10001"

Division kann nur in den Fällen, die "aufgehen", in dieser einfachen Weise programmiert werden.

```
//int2text(0.10,2)
```

Rechnen mit Dualzahlen LEVEL 2

```
div_bin:=(x,y)->(int2text(text2int(x, 2) div text2int(y,2),2)):
div_bin("10110","101")          //Ganzzahlige Division dual
```

"100"

```
mod_bin:=(x,y)->(int2text(text2int(x, 2) mod text2int(y,2),2)):
mod_bin("10110","101")        //Rest bei ganzzahlige Division dual
```

"10"

Wenn man weiß, dass beim Dualbruch die Stellen hinter dem Komma 1/2, 1/4, 1/8,.... bedeuten,

Wenn man weiß, dass beim Dualbruch die Stellen hinter dem Komma 1/2, 1/4, 1/8,.... bedeuten, kann man versuchen, Dualbrüche zu berechnen:
Aufgabe: Bestimme 1/5 als 12-stelligen Dualbruch

```
int2text(floor(1/5 *2^12),2)  // floor rundet immer ab.
```

"1100110011"

Also ist 1/5 = dual 0,001100110011 , es mussten ja 12 Stellen werden

```
int2text(floor(1/5 *2^24),2)  // 1/5 ist ein periodischer Dualbruch!!!!!!
```

"11001100110011100110011"

```
dez
```

65530

Rechnen mit Dualzahlen, Komplementaddition LEVEL 3

Subtrahieren ist gar nicht so ganz einfach, da man die Überträge "leihen" muss.
Das lässt sich nicht gut elementar programmieren. Daher wird auf der elementaren Rechenebene des
Computers "**mit Komplement-Addition subtrahiert".**
Zum Verständnis wird das hier zunächst im Dezimalsystem vorgeführt:

```
523-154 //Rechen Sie das schriftlich von Hand!
```

369

"Minuend - Subtrahend = Differenz"
Rezept für a-b durch Komplementaddition:
Bilde das 9-Komplement zum Subtrahenden. Addiere dieses zum Minuenden.
Fall a>b: Wenn vorn Übertrag 1 kommt, lass die 1 weg.
Addiere 1 zum letzten Ergebnis und du erhältst die Differenz

```
523,9999-154,523+(9999-154),523+(9999-154)-10000,523+(9999-154)-10000+1
```

523, 9845, 10368, 368, 369

```
532-654 //In diesem Fall muss man von Hand andersherum rechnen.
```

– 122

"Minuend - Subtrahend = Differenz"
Rezept für für a-b durch Komplementaddition:
Fall a <b: Bilde das 9-Komplement zum Subtrahenden. Addiere dieses zum Minuenden.
Wenn vorn kein Übertrag 1 kommt, bilde das 9-Komplement vom letzten Ergebnis,
nimm es negativ und du erhältst die Differenz.

```
523,9999-654,523+(9999-654),523+(9999-654)-9999
```

523, 9345, 9868, – 131

```
532-654
```

– 122

Man sieht, dass man der der Subtraktion durch Komplementaddition nicht vorher wissen muss, welche Zahl die größere ist. Für eine Programmierung braucht man lediglich eine Verzweigung nach dem Zwischenergebnis :Summe mit den Neunerkomplement.

Durchführung mit Dualzahlen.
Das 9-Komplement ist nun einfach die Vertauschung von 0 und 1 in der 8-Bit-Darstellung.
Aufgabe: Bilden Sie die Differenz 132- 58 mit Dualzahlen und Komplementaddition:

```
int2text( 132,2),int2text( 58,2), int2text(255-58,2)
```

"10000100", "111010", "11000101"

```
"10000100", "111010", "11000101"
```

```
k_add:=plus_bin("10000100","101010")
```

```
"10101110"
```

```
cc
```

```
cc
```

```
k_add_reduziert:=substring(k_add,1,length(k_add)-1)
```

```
"1010111"
```

```
differenz:=plus_bin(k_add_reduziert,"1")
```

```
"1011000"
```

Probe:

```
plus_bin("10000100","-111010")
```

```
"1001010"
```

Aufgabe: Bilden Sie die Differenz 132- 213 mit Dualzahlen und Komplementaddition:

```
int2text( 132,2),int2text( 213,2), int2text(255-213,2)
```

```
"10000100", "11010101", "101010"
```

```
k_add:=plus_bin("10000100","101010")
```

```
"10101110"
```

Dies sind nun nur 8 Stellen, daher ist das Negative des Komplements die gesuchte Differenz:

```
plus_bin("-11111111","10101110")
```

```
"–1010001"
```

Probe

```
plus_bin("10000100","-11010101")
```

```
"–1010001"
```