restart;

Aufgabe 1: Binärdarstellung

convert(42, *binary*);

101010 **(1.1)**

convert(0.5, *binary*);

0.1000000000 **(1.2)**

convert(420!, *binary*);

(1.3)

```
1101010011110010101011111010001001011010001111000101011101110001100001101\
01
   1111111100111100011001101000101111001000000001100111100100000110010001\
00
   0101111111001110011100101001110001000001100101111000011111010011010010\
00
   0110010000110000011010100101000000011000110100001101000111100100011011\
01
   0101011010110111000001000110111101000010000001101111100111000110110001\
01
   0010111101001001011011101111111110001100000010001101001001001111100010\
11
   1010011000000001000100100001010100010001010110100000100101110011100011\
11
   0000010010001010100000011010011110010001100010011010100001111001011011\
00
   1011111001000001011110011111001000010100111000001100010110100101110101\
11
   0100001011011110000011001010111001010010101111000001110000101010000101\
10
   0011010111011001010110100101000010100111001101011011001001111110011000\
10
   1001101110110111010001010101001111110110111100000100111111010100101111\
01
   1101101100110000000111101001111111011100110100100011001010101000010100\
10
   1101000110001101100100100000111001111000100100110100011110001000001010\
10
   0010100000111111101011000000001001100100110101011110101000001111000001\
10
   1111010000111100011111101001010000110001011111101011101001101011010111\
10
   1111100101000010110111001111001101011011100010110101000100111001111000\
11
   1101111011101100000001101000110111111011100011110000000110100100111010\
00
   0
   1111100101100100000010011010100101010010100111111110001000110001101110\
01
   0011100100101111001111011101110101000010111101010110101110010100000100\
10
   0000010000000110100101001110100001000110011001001011111010010100111111\
11
   1111011000101101100100001100101011000000110111101000011110010000001110\
1
   1110011111111010100110000011001101001101001010011010111111001101001011\
00
   1110010111000001110101100001101110111000100100111001011000000110101001\
11
   0101100101101110101101101010001101100001011110111011101101010001011101\
11
   0011111010000101000100101111011101111000011001100111001000110011010001\
11
   1101101111011011000000011011000011010011011010001100000111100000010001\
10
   0000010000101100101111010101110011001010001011100010000001111111010110\
01
   1111001011101000111010001011110111111100101010101111110111100110110100\
10
   1100011001101100000010000110111110000111100011011001111011010001010011\
11
   1010100100101010100100110000101101110100101001111110111001101010011011\
10
   1000100100111110000110100000100101111011011100011011000101110010001110\
01
   0111101011100111110100010100111101001100011011011111011011011010111101\
01
   0010001110011110000000100110101011100111000111010110011000100011100101\
10
```

```
10101111001100010101001111101110101001010001110001101111011111001001110\
11001010001110111010000010001100101110111111111011101010011101000101100\
11\
10001111010001111010111101000110110001011000110110000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000
convert(-3.75, binary);
                                    -11.11000000                    (1.4)
```

Aufgabe 2: Primzahlen

```
nextprime(10^4);
                                      10007                         (2.1)
nextprime(10^9);
                                    1000000007                      (2.2)
nextprime(10^49);
          10000000000000000000000000000000000000000000009          (2.3)
```

Aufgabe 3: Prozeduren

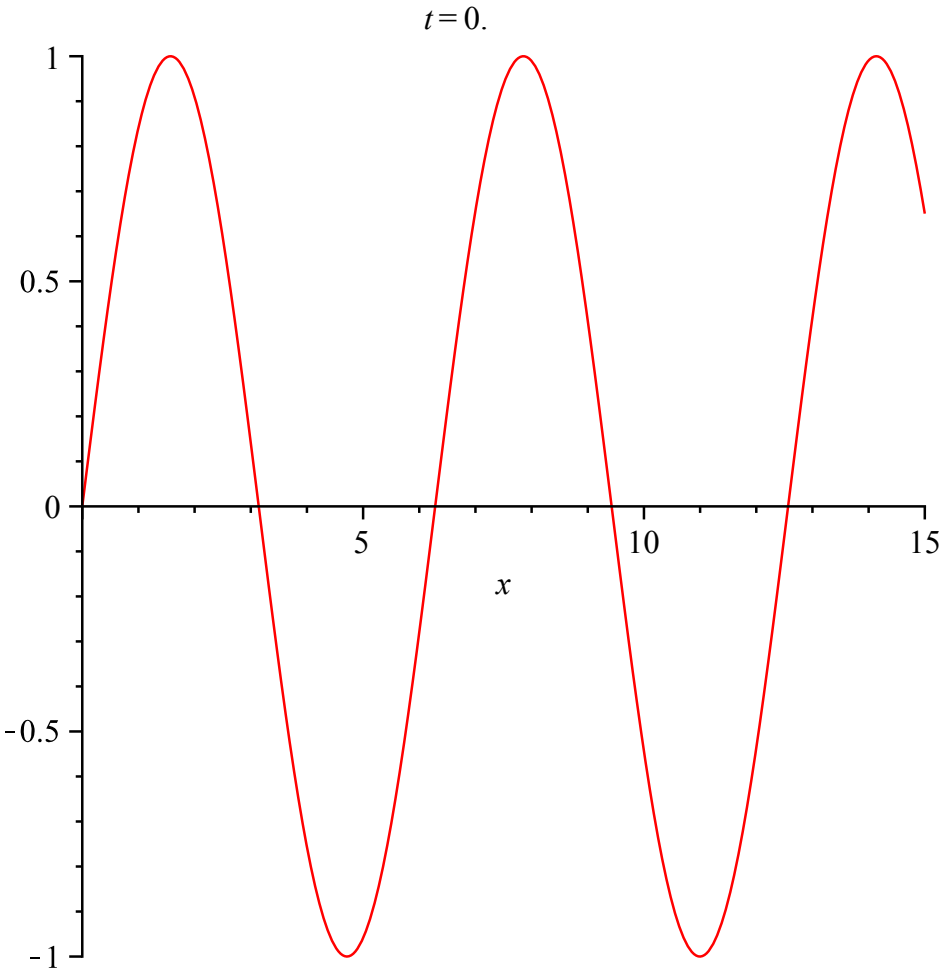
```
p := proc(n :: nonnegint)
  if n < 10 then return "Zahl ist einstellig";
  elif n < 100 then return "Zahl ist zweistellig";
  elif n < 1000 then return "Zahl ist dreistellig";
  else return "Zahl hat mehr als drei Stellen";
  end if;
end proc;
```

```
proc(n::nonnegint)                                                    (3.1)
    if n < 10 then
        return "Zahl ist einstellig"
    elif n < 100 then
        return "Zahl ist zweistellig"
    elif n < 1000 then
        return "Zahl ist dreistellig"
    else
        return "Zahl hat mehr als drei Stellen"
    end if
end proc
```

```
p(0);
                                  "Zahl ist einstellig"               (3.2)
p(9);
                                  "Zahl ist einstellig"               (3.3)
p(10);
                                  "Zahl ist zweistellig"              (3.4)
p(99);
                                  "Zahl ist zweistellig"              (3.5)
p(100);
                                  "Zahl ist dreistellig"              (3.6)
p(999);
                                  "Zahl ist dreistellig"              (3.7)
p(1000);
                          "Zahl hat mehr als drei Stellen"            (3.8)
p(42!);
                          "Zahl hat mehr als drei Stellen"            (3.9)
```

Aufgabe 4: Animation

```
with(plots):
animate(plot, [sin(x+t), x=0..15], t=0..4·π);
```




```
# Animation startet z.B. durch Rechtsklick - Animation - Play
```

Aufgabe 5: Stückweise Definition einer Funktion

$$f := x \to piecewise\left(-5 \le x < -3, x+5, -3 \le x < 3, 1+\frac{1}{9}\cdot x^2, 3 \le x < 5, 5-x, 0\right);$$

$$x \to piecewise\left(-5 \le x \text{ and } x < -3, x+5, -3 \le x \text{ and } x < 3, 1+\frac{1}{9}\,x^2, 3 \le x \text{ and } x < 5, 5-x, 0\right) \quad \textbf{(5.1)}$$

$$plot(f(x), x = -6..6, scaling = constrained);$$