1) Print even numbers from 6 to 10.

```
> for i from 6 by 2 to 10 do print(i) end do;
```

2) Find the sum of all two-digit odd numbers from 11 to 99.

```
> mysum := 0 :
  for i from 11 by 2 while i < 100 do
     mysum := mysum + i;
     #print(mysum);
   end do: #a ; instead a : leads to different outputs
   mysum;
```

3) Multiply the entries of an expression sequence.

```
> restart;
   total := 1 :
  for z in 1, x, y, q^2, 3  do
     total := total·z
   end do:
   total;
   x := 2 :
   q := 3 :
  total;
```

3) Add together the contents of a list.

```
> ?cat
> restart;
  y := 3;
  myconstruction := "";
  for z in [1, "+", y, "·", "q^2", "·", 3] do
     myconstruction := cat(myconstruction, z)
   end do
  myconstruction;

> ?parse

> q := 4;
> qq := parse(myconstruction);
> qq;
```

Procedures

Flow control constructions, simple commands and comparison operators can be bound together; in a so called
procedure. The simplest possible procedure looks as follows.

```
proc(parameter sequence)
   statements;
end proc:
```

```
> restart;
  myfactorial := proc(n)
      local r, i;
      r := 1;
      for i from 1 by 1 to n do
         r := r · i;
```

```
        print(r);
      od;
      return r;
    end proc;
> myfactorial(4);
```

Maple allows recursive procedure calls:

```
> restart;
  myfactorial2 := proc(n)
    if (n < 2) then return 1
    else return n·myfactorial2(n − 1);
    fi;
  end proc;
> myfactorial2(4);
```

Functional-Operators

Maple allows the definition of so called functional operators.

- A functional operator in Maple is a special form of a procedure. Functional operators are written using arrow notation.

  vars -> result

  Here, vars is a sequence of variable names (or a single variable) and result is the result of the procedure acting on vars.

- For example, the following

  x -> x^2

  represents the function that squares its argument.

- Multivariate and vector functions are also allowed. You must put parentheses around vars or result whenever they are expression sequences. For example, the following functions have the correct syntax.

  (x,y) -> x^2 + y^2
  x -> (2*x, 3*x^4)
  (x,y,z) -> (x*y, y*z)

```
> restart; #examine the differences between functions and expressions
> f := x → x^4 − 3·x + 21;
> f(3);
> g := x^4 − 3·x + 21;
> eval(g, x = 3);
> h1 := 2·f;
> h1(2);
>
> h2 := 2·g;
> h2(2);
> eval(h2, x = 2);
>
> x := 5; simplify(h2);
> h2;
>
```

```
>
>
```

The Maple Library

The Maple library consists of for parts:

• the standard library
• the update library
• packages
• share library (user-contributed)

Until now, we only used commands and operations from the standard- and the update library.

However: There are so called packeges for more specialized purposes in Maple,
e.g. the LinearAlgebra package for matrix-vector computations or the numtheory-package.
Functions from those packages can be used with the following syntax:

PackageName[FunctionName](FunctionParameters)

Here two examples:

```
> restart;
```

> $A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$

```
> LinearAlgebra[Transpose](A); # transposes the matrix A
> numtheory[divisors](68); # prints the divisors of 68 to the screen
```

Often, you want to use a package more intensively. Then you can abbriviate the package-commands with the with()-command:

```
> with(LinearAlgebra);
> with(numtheory);
> Transpose(A);
> factorset(96);
```

Solving Equations

Examples:

```
> restart;
```

> $x + y = 23;$

> $eq1 := (x^3 - 2 \cdot x^2 + 23 \cdot x - 108 = 0);$

> $eq2 := \left(2 \cdot x + 4 \cdot y = \frac{29}{6}\right);$

```
> res := simplify(solve(eq1, x)[1]);
> fsolve(eq1, x);
> ?fsolve
> fsolve({eq1, eq2}, {x, y});
> solve({eq1, eq2}, {x, y});
> evalf(res);
```

```
> evalf(res[1]);
> reslist := convert({res},'list'); # also possible: reslist := [res];
> reslist2 := [res];
> evalf(reslist[1]);
>
> solve({x = reslist[1], eq2}, {x, y});
> fsolve({x = reslist[1], eq2}, {x, y});
>
```

A further example:

```
> factorial(10000)/factorial(9999);
> solve(x·factorial(9999) = 10000, x);
>
```

Sequences, Limits and Series

Little dictionary:
limit : Grenzwert
sequence : Folge
series : Reihe

Definition (*sequence*): A **sequence** of real numbers is a mapping from ℕ → ℝ.
Example: Let a_n:= 1/n, n ⩾1. This gives the sequence (1, 1/2, 1/3, ...)

Definition (*convergence,limit*): Let $(a_n)_{n \in \mathbb{N}}$ be a sequence of real numbers. A sequence is called convergent towards a ∈ℝ, if and only if:
For all $\epsilon > 0$, it exists an N(ϵ) ∈ ℕ such that
$|a_n - a| < \epsilon$ for all $n \geq N(\epsilon)$.
We write $\lim_{n \to \infty} a_n$ = a.

Definition (*series*): Let $(a_n)_{n \in \mathbb{N}}$ be a sequence of real numbers. The sequence $s_n := \sum_{k=0}^{n} a_k$, $n \in \mathbb{N}$ of sums is called **series**, and is described with the help of $\sum_{n=0}^{\infty} a_n$.

Definition (*absolute convergence*): A series $\sum_{n=0}^{\infty} a_n$ is said to **converge absolutely** if the series

$$\sum_{n=0}^{\infty} |a_n|$$

converges, where $|a_n|$ denotes the absolute value of a_n.

Definition (*limits at functions*): Let f : D ⟶ ℝ a real valued function on the domain D ⊆ℝ with a point
a ∈ℝ, such that there exists at least one sequence $(a_n)_{n \in \mathbb{N}}$, $a_n \in$ D
with

$\lim_{n \to \infty} a_n$ = a.
We write
$$\lim_{x \to a} f(x) = c$$
if and only if it is valid:
$\lim_{n \to \infty} f(x_n)$ = c for all $(x_n)_{n \in \mathbb{N}}$ with $\lim_{n \to \infty} x_n$ = a.

```
> restart;
>
```

Computations of limits:

```
> plot(1/x · sin(x), x = 1 ..20);
>
> l2 := [seq([n, 1/n · sin(n)], n = 1 ..20)];
  plot(l2, x = 0 ..15, style = point, symbol = circle);
> plots[pointplot]({seq([x, 1/x · sin(x)], x = 1 ..20)});
> limit(sin(x), x = 0);
> limit(sin(x)/x, x = 0);
> plot([sin(x), x], x = -3.5 ..3.5, thickness = 2);
> plot(signum(x), x = -1 ..1);
> limit(signum(x), x = 0);
> limit(signum(x), x = 0, left);
> limit(signum(x), x = 0, right);
> limit(exp(x), x = ∞);
```

Further examples.

```
> limit(n^2/(n^3 + 1), n = ∞);
> limit((π ·n^3 + 17 · n + n)/(n^3 + 39), n =∞); # wrong space!
> limit(n^k/n!, n = ∞);
> limit(n^n/n!, n = ∞);
> limit(n^k/n!, n = 0);
> limit(n^k/n!, n = 0) assuming k > 0;
> limit(n^k/n!, n = 0) assuming k < 0;
```

> ?factorial

> $evalb(99! = \Gamma(100));$

> $simplify(n! - \Gamma(n+1));$

> $plot\left(\frac{n^{-2}}{n!}, n = -0.00001..0.00001\right); evalf\left(\frac{0.00001^{-2}}{0.00001!}\right);$

> $limit\left(\frac{n^k}{n!}, n = 0\right)$ assuming $k = 0;$

> $eval\left(limit\left(\frac{n^k}{n!}, n = 0\right)\right)$ assuming $k = 0;$

> $limit\left(\frac{n^0}{n!}, n = 0\right);$

>

Series

First idea

> $restart;$

> $sum(a[k], k = 0..\infty);$

> $plot\left(\sum_{i=0}^{n}\left(\frac{1}{2}\right)^i, n = 0..10\right);$

> $sum\left(\left(\frac{1}{2}\right)^n, n = 0..\infty\right);$ *#is the same as:*

> $limit\left(\sum_{i=0}^{n}\left(\frac{1}{2}\right)^i, n = \infty\right);$ *# is the same as:*

> $\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i;$

Is $\frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}}$ *equal* **to** $\sum_{i=0}^{n}\left(\frac{1}{2}\right)^i$?

> $f := \frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}} - \sum_{i=0}^{n}\left(\frac{1}{2}\right)^i;$

> $\sum_{i=0}^{\infty} x^i;$

> *#computing with series*

> $\sum_{i=0}^{12} x^i + \sum_{i=15}^{\infty} x^i;$

> $simplify(\%);$

Harmonic Series

> $Harmonic := \sum_{i=1}^{\infty} \frac{1}{i};$

> $\sum_{i=1}^{\infty} (-1)^i \frac{1}{i};$

> $AlternatingHarmonic := n \to \frac{(-1)^n}{n};$

> $map(AlternatingHarmonic, [seq(1..10)]);$

> $sum(AlternatingHarmonic(n), n = 1..\infty)$

The Riemann Series Theorem

Theorem: Let f $: \mathbb{N} \to \mathbb{R}$ be a sequence such that the series $\sum_{k=1}^{\infty} f(k)$ converges but not absolutely.

Then: For each real x there is a bijection (a re-ordering) $\beta: \mathbb{N} \to \mathbb{N}$ such that $\sum_{k=1}^{\infty} f(\beta(k)) = x.$

We want to construct such a reordering for given f and x. First we need two short functions which will be helpful.

```
> restart;
> AlternatingHarmonic := n -> (-1)^n / n;
>
> FindNextPositiveStartingIndex := proc(f, k)
    local i;
    i := k;
    while f(i) <= 0 do
     i := i + 1
    end do;
    return i
  end proc;
> FindNextNegativeStartingIndex :=
  proc(f, k)
     local i;
     i := k;
     while 0 <= f(i) do
        i := i + 1
     end do;
     return i;
  end proc
> AlternatingHarmonic(2);
> FindNextPositiveStartingIndex(AlternatingHarmonic, 1);
>
>
> Riemann := proc(f, x, k)    # here: only vor strictly alternating series
```

```
        local s, pix, nix, j, p, n, ret;
        s := 0;
        ret := -1;
        pix := FindNextPositiveStartingIndex(f, 1);
        nix := FindNextNegativeStartingIndex(f, 1);
        for j from 1 to k do
          if evalf(s) < evalf(x) then
            s := s + f(pix);
            ret := f(pix);
            pix := pix + 2;
          else
            s := s + f(nix);
            ret := f(nix);
            nix := nix + 2;
          end if;
        end do;
        return [ret, evalf(s)]
    end proc
```

> $seq\big(Riemann\big(AlternatingHarmonic, \pi, i\big)[2], i = 1000..1020\big);$

> $seq\left(Riemann\left(AlternatingHarmonic, \frac{100}{1001}, i\right)[2], i = 1000..1020\right); \frac{100.0}{1001.0};$

> $evalb(0 < sqrt(2)); evalb(0 < evalf(\mathrm{sqrt}(2)));$

> $sum(AlternatingHarmonic(n), n = 1..\infty);$

> $rseq := seq\left(\left[i, Riemann\left(AlternatingHarmonic, \frac{1000}{1001}, i\right)[2]\right], i = 1..400\right):$

>

> $plots[pointplot]\left(\left\{seq\left(\left[x, \frac{1000}{1001}\right], x = 1..400\right), \mathrm{rseq}\right\}\right);$

>

Points, Vectors, and Matrices

> $with(LinearAlgebra); with(plots);$

Let us inspect (column) vectors.

> $p := \langle 0, 1\rangle; r := \langle 1, 2\rangle;$

> $p[1];$

> $r[2];$

> $l := p + \lambda \cdot r;$ *# this is one posibility to encode a line*

Now, we want to compute the shortest distance from point q := <2,1> to the line.

> $q := \langle 2, 1\rangle;$

> $lineplot := plot([l[1], l[2], \lambda = -2..2]);$ *# a so called parametric plot*

> $display(lineplot);$

> $f := \lambda \to p + \lambda \cdot r;$ *# the line, encoded as a function in Lambda*

> $s := seq([l[1], l[2]], \lambda = -2..2);$ *# some points on our line with the help of the expression l*

> $t := seq\left(\left[f\left(\frac{x}{10}\right)[1], f\left(\frac{x}{10}\right)[2]\right], x = -20..20\right);$
 # some further points on the line with the help of function f

> $pointline := pointplot([t]);$ *# a plot is a Maple-object, not the graphical output*

```
> Qplot := pointplot(q);
> a1 := arrow([0, 0], p, width = [0.075, relative = false], head_length = [0.4, relative
      = false], color = green);
> #a2 := arrow(p, 0.3· r, width = [0.075, relative = false], head_length = [0.4, relative
      = false], color = blue);
> a2 := arrow(p, DotProduct(q - p, r)/DotProduct(r, r) · r, width = [0.075, relative = false], head_length
      = [0.4, relative = false], color = blue);
> HitPoint := subs(λ = DotProduct(q - p, r)/DotProduct(r, r), l);
      # one possibility to compute the point on line l which is nearest to point q.
> λ_hit := DotProduct(q - p, r)/DotProduct(r, r); #another possibility follows:
> HitPoint2 := λ_hit·r + p;
> DotProduct(p + λ_hit·r-q, r);
> display([a1, a2, lineplot, pointline, Qplot, pointplot([q, HitPoint], connect = true, thickness
      = 1, linestyle = dash)], view = [-2.5 ..2.5, -2 ..2]);
>
> myDotProduct := proc(u, v, k)# what is a DotProduct?
    local j, res, lu, lv;
    res := 0;
    for j from 1 to k do
      res := res + u[j]·v[j];
    end do;
    #return res;
    lu := convert(u,'list');
    lv := convert(v,'list');
    return sum(lu[i]·lv[i], i = 1..k);
  end proc:
> DotProduct(q - p, r)/DotProduct(r, r);
> myDotProduct(q - p, r, 2)/myDotProduct(r, r, 2); # last parameter gives the dimension

>
>
```