▪ Introduction to *Mathematica.* Volume II - Programming

download this document from `www.hakenberg.de`

⚠ Please **deactivate NUM-Lock**.
⚠ Proceed by copying the expressions into *Mathematica*. Press SHIFT RET together to **evaluate** the expressions. Vary the input.
⚠ When you recieve weird error messages upon evaluation, although your input seems correct, use the menu item **Kernel** → **Quit-Kernel** to reset the memory of *Mathematica* and start evaluation from the beginning.

Difference between := and =

```
rnd1 = Random[]
rnd2 := Random[]
{rnd1, rnd1, rnd2, rnd2}
```

Various ways to define **functions** and to **call** them with arguments
💡 y_List requires the second argument to be a list. z_: *i* indicates, that the third argument *z* is optional and the default value is *i* in this particular case. Produce ⊗ with ESC c* ESC

```
myF1[x_, y_List, z_: ⅈ] := y^x + z
myF2 = #2^#1 + #3 &
X_ ⊗ Y_ := Outer[Times, X, Y]

myF1[3, a]
myF1[3, {a, b}]
myF2[3, a, ⅈ]
{1, 2, 3} ⊗ {a, b, c, d} // MatrixForm

a_##1 &[4, x, π]
```

Instead of using "[...]" for function-calls, alternative notation prooves to be extremly useful.
⚠ when you are familiar to the syntax below, set `myF` equal to functions like `Sin`, `Plus`, etc.

```
hisF@yourF@myF@3.
{1, 2, 3.} // myF
myF @@ {1, 2, 3.}
myF /@ {1, 2, 3.}
myF //@ {1, 2, 3.}
Nest[myF, x, 6]

DSolve @@ {{y'[x] == y[x] + x, y[0] == α}, {y[x]}, x}
#1 + Cosh[#2^#3] & @@ {x, y, z}
```

The following code implements the formula that alternates/skews a given tensor of arbitrary rank. Let X be a tensor of rank r on say $\mathbb{R}^n$, then X consists of n^r entries addressed by $X_{i_1,\dots,i_r}$, where $i_j \in \{1, \dots, n\}$. With $\mathcal{S}_r$ as the group of permutations of r elements, the alternate tensor of X is defined

$$\mathcal{A}(X)_{i_1,\dots,i_r} = \frac{1}{r!} \sum_{\sigma\in\mathcal{S}_r} \mathrm{sign}(\sigma)\cdot X_{\sigma(i_1,\dots,i_r)}$$

Note, how the different functional operators are combined below to produce a compact appearance.
💡 $\mathcal{A}$ corresponds to ESC scA ESC, in which "sc" stands for *script*. Try also ESC goA ESC.

```
rank = TensorRank
𝒜[X_] :=
 1 / rank[X] ! Plus @@ (Signature[#1] Transpose[X, #1] & /@ Permutations[Range[rank[X]]])
𝒜[( x  4
    π  y )] // MatrixForm
```

The following functions help to **circumvent while/for** loops.
⚠ read about `FixedPoint` before you modify the code below.

```
Fold[myF, a0, {a1, a2, a3}]
FoldList[myF, a0, {a1, a2, a3}]
NestList[Min[5, #1 + Random[]] &, 0, 6]
FixedPointList[Min[5, #1 + Random[]] &, 0]
FixedPoint[Min[5, #1 + Random[]] &, 0]
```

Memo functions help you to increase efficiency. The code below defines a recursive computation of n factorial, i.e. $n!$. *Mathematica* creates a lookup table for all input parameters that `fac` has seen so far.

```
fac[0] = 1;
fac[n_] := fac[n] = n fac[n - 1];
fac[5]
? fac
```

`If` and `Which` are classical commands to **control the flow** of the program.
💡 for ≠ just type `!=` ✍ try different values for `n` in the last line

```
If[#1 ≠ 0, 1 / #1, ∞] & /@ {1, 1 / 2, 3, 0}
"we" // Which[#1 == "I", "am", #1 == "he", "is", True, "are"] &
n = 4; While[Mod[n, 5] ≠ 1, n = Mod[n^3 - 2 n + 1, 11]; Print[n]]
```

Get familiar to some useful functions that **manipulate lists**.

```
myList = {a1, {a2, a3}, a4, {{a5}, a6}};
Length[myList]
Flatten[myList]
Reverse[myList]
Drop[myList, 1]
Range[4, 10]
Select[{20, 2, x, 4, π, 10}, #1 > π &]
```

A **Module** is the most general form of a function/procedure in *Mathematica* and only useful, when a lot of decisions and repetitions are required. The code below controls, i.e. accelerates, a particle on a line from initial state ($\mathbf{vel_0}$, $\text{pos}_0 = 0$) to the target state ($\mathbf{vel}_T$, $\mathbf{pos}_T$). The bold variables are input parameters to the program. Acceleration is bounded and the control is *time optimal*. The particle is subject to friction. For higher dimensions this problem is hard!
✍ The algorithm as shown could be stable, however, if `vel=0` there are difficulties in computing `est`, which is easily traced back to the computation of `sgn` one line above. Lookup the definition of `Sign` and make a minor change to the algorithm by treating the case `vel=0` differently. Launch the new code with `move[0,-1,.5]`.

```
move[vel0_, posT_, velT_] :=
  Module[{cnt, states = {}, pos = 0, vel = vel0, tmp, τ = .01, est},
   For[cnt = 0, cnt < 1500, cnt ++;
    sgn = Sign[-vel];
    est = vel - velT + sgn Log[(sgn - vel) / (sgn - velT)];
    acc = If[pos + est < posT, 1, -1];
    AppendTo[states, {vel += acc τ - vel τ, pos += vel τ}]
   ];
   ListPlot[states, AxesLabel → "Phase", PlotRange → All];
  ];
move[-.7, 1, -.4]
```

Functions like `Sin` and `Exp` have the **attribute** `Listable`, see example below. Check out the attribute `Orderless`.

```
SetAttributes[myZ, {Listable}]
myZ[{1, {2, 3}, 4}]
```

⚠ We close with an example for students familiar to lie algebras: The following lines lists all possible commutator tables of three dimensional lie algebras. For certain choices of coefficients $a_{i,j,k}$ they are necessarily isomorphic. The symbol ⋆ dumps upon ESC*5ESC.

```
ad⋆[n_] := Array[Which[#2 > #3, a#1,#2,#3, #2 < #3, -a#1,#3,#2, True, 0] &, {n, n, n}];
adJacobi[ad_] :=
  Plus @@ (Transpose[ad.ad, #1] & /@ {{1, 2, 3, 4}, {1, 3, 4, 2}, {1, 4, 2, 3}});
ShowAd[ad_] := MatrixForm[Array[e#1 &, {Length[ad]}].-ad];
ShowAd[Simplify[ad⋆[3] /. #1]] & /@ Solve[0 == #1 & /@ Flatten[adJacobi[ad⋆[3]]]]
```