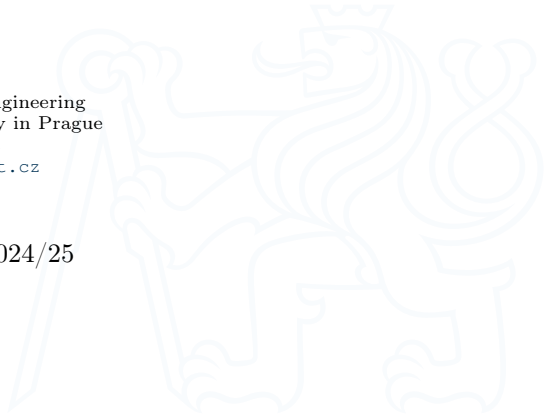
Lecture 11: Expressions in general.

A8B17CAS

Jozef Lukáč

Department of Radio Engineering
Czech Technical University in Prague
Czech Republic
lukacjo1@fel.cvut.cz

December 2
Winter semester 2024/25



Outline

(De)Composition of Expressions.

- Unifying idea of MATHEMATICA:
  - *Everything can be represented as a symbolic expression.*
- Examples of equivalent expressions:

`x+y+z`	`Plus[x,y,z]`
`x y z`	`Times[x,y,z]`
`x^n`	`Power[x,n]`
`{a,b,c}`	`List[a,b,c]`
`a->b`	`Rule[a,b]`
`a=b`	`Set[a,b]`
`x==Sin[x]`	`Equal[x,Sin[x]]`
`p && !q`	`And[p,Not[q]]`
`m[[1]]+=a`	`AddTo[Part[m,1],a]`

(De)Composition of Expressions.

- General form of any expression: `f[x,y,...]`
- Some interpretations of parts of expressions.

meaning of f	*meaning of x, y*	*examples*
Function	arguments or parameters	`Sin[x]`, `f[x,y]`
Command	arguments or parameters	`Expand[(x+1)^2]`
Operator	operands	`x + y`, `a = b`
Head	elements	`{a, b, c}`
Object type	contents	`RGBColor[r,g,b]`

- Atomic types of objects used in expressions:

`Symbol`	symbol (extract name using `SymbolName`)
`String`	character string *e.g.* `"cccc"` (extract characters using `Characters`)
`Integer`	integer (extract digits using `IntegerDigits`)
`Real`	approximate real number (extract digits using `RealDigits`)
`Rational`	rational number (extract parts using `Numerator` and `Denominator`)
`Complex`	complex number (extract parts using `Re` and `Im`)

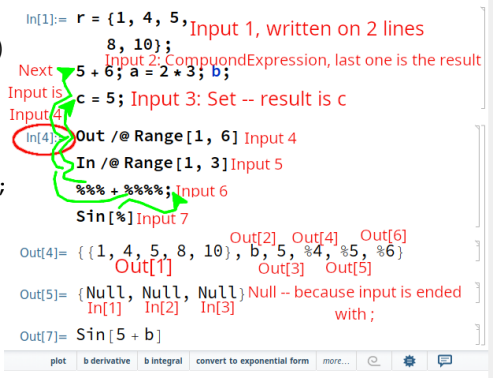
Types of Tokens in MATHEMATICA.

- types of tokens in Mathematica:

symbols	`a`, `xyz`, $\alpha\beta$, γ
strings	`"some text"`, `"`α + β`"`
numbers	`123.456`, `3*^45`
operators	`+`, `->`, `!=`
input to be ignored	`(* comment *)`

History of evaluated expressions.

- Note the **In** and **Out** symbols at the beginning of rows
- Every input expression (usually written on a single row) sent to kernel is indexed by **In[i]**. Index is shown just for the first expression in an input cell
- For every input in **In[i]**, there is an **Out[i]**. **Out[i]** is shown only if input is not terminated by the semicolon **;**
- User can access the output of previously evaluated cells (the **Out** cells), even if their output is suppressed by the semicolon **;**.
- Access by a relative index:
  - **%** – previous output cell,
  - **%%** – 2-nd output cell from the end,
  - **%%%** – third output cell from the end,
- Access (**Out**put expression) by an absolute index: **%***n* (*e.g.* **%12**)



History of evaluated expressions.

- Rewrite the following code without using names of variables (`x`, `s`, `y`, `z`, `matM`)

```
x = 5
s = Cos[x]^2
s - x
y = Range[0, 1, 0.5]
z = Range[-2, 3, 1]
matM = Outer[Plus, y, z]
Mean /@ matM
```

Exercise

Ways to Input Expressions.

- Same thing can be expressed by different ways even in the same language.

`f[x,y]`	standard form for `f[x,y]`
`f @ x`	prefix form for `f[x]`
`x // f`	postfix form for `f[x]`
`x~f~y`	infix form for `f[x,y]`

- Examples:

```
x + y//f                                  f[x + y]
3^(1/4) + 1//N                            2.31607
{a,b,c}~Join~{d,e}                        {a, b, c, d, e}
```

Ways to Input Expressions.

- Write `Sin[x]` in the prefix and postfix form.
  ...
  ...
- Write `x < y` in the standard and infix form.
  ...
  ...
- Evaluate 5^{100} numerically with precision 20 digits (use `N` command), express the evaluation in the standard and infix form.
  ...
  ...

Exercise

Forms of Expressions.

- MATHEMATICA can accept input and return output in different forms (Input and Output forms).
- Some examples of "Forms":

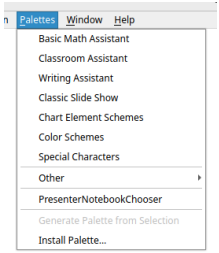
  InputForm, **OutputForm**, **StandardForm**, **TraditionalForm**, **MatrixForm**, **TreeForm**, **FullForm**, **TableForm**

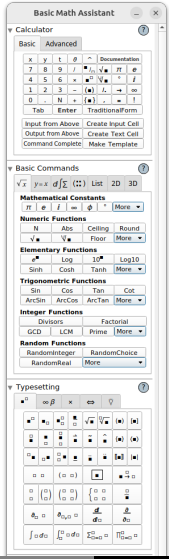
- *E.g.*

```
expr = {{2,3},{4,7}};
expr//InputForm
expr//OutputForm
expr//StandardForm
expr//TraditionalForm
expr//MatrixForm
expr//TreeForm
expr//FullForm
expr//TableForm
```

Input of Special Characters.

- In notebooks, we can also write special characters, such as greek letters (*e.g.* α,τ).
  - type *ESC* + *name of the character* + *ESC*
  - *E.g.* `ESC gamma ESC`, resp. `\[Gamma]` is γ
- Also, special mathematical notation (*e.g.* upper index, ...) is supported.
  - *E.g.* `Ctrl 2 4`, resp. `Ctrl @ 4` is $\sqrt{4}$.
- Or open a pallet with symbols ...

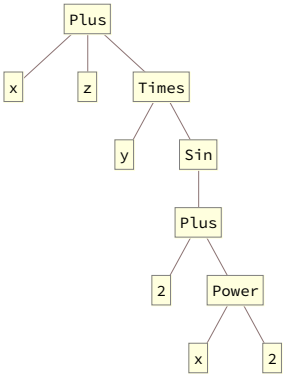




Properties of Expressions, Indexations.

- Every expression can be represented by a tree (see. **TreeForm** of expression)
- *E.g.*

```
expr2=x+y*Sin[2+x^2]+z;
expr2//TreeForm
```



- Properties of Expressions:
  - **Head** of expression – **Plus**.
  - **Level** of expression – *e.g.* **Level[expr2,1]**.
  - **Length** of expression – number of arguments at the first level.
  - **Depth** of expression – depth of the "expression tree" – number of levels.
- Indexation in Expressions:
  - We can specify a part(s) of expression by *expr[[part specification]]*, resp. by **Part** command. *E.g.*

```
expr2[[1]]
expr2[[3]][[2]] (*,resp.  expr2[[3, 2]]*)
expr2[[3, 2, 0]]
mat = {{-1, 1, 0, 2, 6, 10, 22}, {5, 3, 1, -3,
    13, 8, 1}, Range[-2, 4]}
mat[[2, All]]
mat[[;; , 2]]
mat[[;; , 1 ;; 6 ;; 2]]
mat[[3, {5, 6}]]
```

Properties of Expressions, Indexations.

- We will read a table of data from a file. Execute the following commands.

```
SetDirectory[NotebookDirectory[]<>"data"];(* set curr. dir. to 'ntb. dir'/data *)
FileNames[] (* list of filenames in the current directory *)
tab01 = Import["rec01_containers.txt", "Data"]
```

- Record 01 contains records about containers (name/label on it, volume of water in it, and the weight of the empty container). Determine *mean volume* of water in containers and *maximal weight* of empty container.

  ...
  ...

- Read record 02 (`"rec02_numbers.txt"`). From the given matrix, extract a submatrix such that it contains the 4-th, 5-th row and 1-st, 3-rd, and 5-th column of the original matrix.

  ...
  ...

Exercise

Properties of Expressions, Indexations (2).

- Extract a subexpression/subpolynomial from a polynomial `pol` (a Taylor series expansion of $e^{\cos(x)}$ at $x = 0$). The subpolynomial should include $x^4, \ldots, x^8$. Use indexing in `pol`.

```
Exp[Cos[x]]
Series[%, {x, 0, 10}];
pol = % // Normal // Expand
...
```

- Assume you have an expression expr $= 5^{\tan(x^2)-1} + 4\,e^{3\,\cos^2(x+3)}$. Display the expr using `TreeForm`. Index in the expression `expr` to get a) x^2, b) $\cos(x+3)$ c) 5

```
expr = 5^(Tan[x^2] - 1) + 4 E^(3 Cos[x + 3]^2);
expr[[ ... ]] (* a) *)
expr[[ ... ]] (* b) *)
expr[[ ... ]] (* c) *)
```

Exercise

Simple Rules.

- Rule says MATHEMATICA to take some part of an expression and *substitute* it for another one (see **Rule**, **ReplaceAll**, resp. posfix form **/.**, **ReplaceRepeated**, with postfix form **//.**, and **Replace**).
- *E.g.*
```
expr = a + b^2 + a c;
rule = a -> 3;
ReplaceAll[expr, rule] (* resp. expr /. rule *)
```
- We can apply multiple rules, *e.g.*
```
rule2 = b->-5;
ReplaceAll[expr, {rule,rule2}] (* resp. expr /. {rule, rule2} *)
```
- Note the difference for nested rules.
- *E.g.* `expr/.{{a->4},{a->5}}`
- Even heads of expressions can be replaced.
- *E.g.* `Sin[x^3 + 4] + Tan[y/4]/. {Plus -> Power, Sin -> Cos}`
- Solution of equation(s) are returned as a list of (nested) rules.
- *E.g.*
```
Clear[x]; sols = Solve[x^3 + x + 1 == 0, x] // N
x /. sols (* resp. ReplaceAll[x, sols]  ...or x ~ ReplaceAll ~ sols *)
```

Simple Rules.

- Solve the equation $x^2 + 4x - 1 = 0$ for x, use `Solve`, or `NSolve`. Extract values of x from the solution-rules. (Hint: use `ReplaceRepeated`, resp. `/.` with `x` expression and the solution-rules).

  ...
  ...

- Solve the system of equations

$$x^2 + xy - 1 = 0,$$
$$y^2 + x + 2 = 0,$$

  for x, y and extract the solution pairs `{x,y}`. (Hint: use `ReplaceRepeated`, resp. `/.` with `{x,y}` expression and the solution-rules).

  ...
  ...
  ...

Exercise

Apply and Map.

- MATHEMATICA supports concept known as functional programming.
- Basic examples of it are commands **Map** (resp. infix form **/@**) and **Apply** (resp. infix form: **@@**).
- **Map** takes a head (*e.g.* **Plus**, **List**, **f3**, ...) and "*wraps it around*" all arguments in the given expression, *e.g.*

```
myList={x^2,5,Sin[11x]};
Map[f3,myList]
```

→ `{f3[x^2],f3[5],f3[Sin[11x]]}`

- Also, we can specify the *"mapping level"*, *e.g.*

```
TreeForm @ myList (*to see levels of myList*)
Map[f3,myList,{2}]
```

→ `{f3[x]^f3[2], 5, Sin[f3[11 x]]}`

- **Apply** takes a head (*e.g.* **Plus**, **List**, **f3**, ...) and *substitutes* head of the input expression, *e.g.* **Apply[f3,myList]**
  →`f3[x^2, 5, Sin[11 x]]`
- Also, we can specify the *"application-level"*, *e.g.* **Apply[f3,myList,{2}]**
  →`{x^2, 5, Sin[f3[11, x]]}`
- Some examples:

```
myList={x^2, 5, Sin[11x]};TreeForm@myList
Map[f3, myList] (*resp. f3/@ myList*)
Map[f3, myList, {2}]
Apply[f3, myList] (*resp. f3 @@ myList*)
Apply[f3,myList,{1}] (*resp. f3@@@myList*)
Apply[f3, myList, {2}]
v = {5, 4, 6};
Apply[Times, v]
Plus @@ v
mat={{0,1,-3},{5,4,2}};
Apply[Plus,%,{1}]
Map[Total,%%]
```

Apply and Map.

Assume a 2 x 3 matrix **mat** and its transpose **matT**.

1. Compute the dot product of the two rows of **mat**, (use **Dot**).
2. Use the command **Outer** with the first and the second row of the matrix **mat**, (**mat[[1]]** and **mat[[2]]**) to create a 3D matrix **mat3D**.
3. Compute product of the 2-element lists in **mat3D**
4. Create 3D matrix from **mat**, such that each matrix element is "wrapped" by a list: **i -> {i}**
5. Compute sum of each row of **matT**. Implement 2 solutions: one using **Total**, the second using **Plus**

```
mat = {{-2, 1, -3}, {5, 4, 2}}
matT = Transpose@ %
... (*Dot*)
mat3D =
Outer[..., mat[[1]], mat[[2]]] (*Outer*)
{% // Grid , % //MatrixForm}(*Grid-view and MatrixForm
    from the previous result*)
... (*Times*)
% // Grid (*Grid-view from the previous result*)
... (* i -> {i} *)
... (*Total*)
... (*Plus*)
```

```
Out[1]= {{-2, 1, -3}, {5, 4, 2}}   mat
Out[2]= {{-2, 5}, {1, 4}, {-3, 2}} matT
Out[3]= -12 Dot of the two rows of mat
Out[4]= {{{-2, 5}, {-2, 4}, {-2, 2}}, {{1, 5}, {1, 4}, {1, 2}}, {{-3, 5}, {-3, 4}, {-3, 2}}}
                                    mat 3D created using Outer ...
Out[5]= { {-2, 5}  {-2, 4}  {-2, 2}      ( (-2)  (-2)  (-2) )
          {1, 5}   {1, 4}   {1, 2}  ,    ( (1)   (1)   (1)  )
          {-3, 5}  {-3, 4}  {-3, 2}      ( (-3)  (-3)  (-3) )
                                         }  Grid -view and MatrixForm
                                            of the mat3D
Out[6]= {{-10, -8, -4}, {5, 4, 2}, {-15, -12, -6}}  matrix of products
        -10  -8  -4  Grid - view of
Out[7]=   5   4   2  the matrix of
        -15  -12  -6  products
Out[8]= {{{-2}, {1}, {-3}}, {{5}, {4}, {2}}} i -> {i}
Out[9]= {3, 5, -1} sum of rows of matT using Total
Out[10]=
        {3, 5, -1}  sum of rows of matT using Plus
```

Exercise

Pure Functions + Simple Functions

- In MATHEMATICA, we can define functions in several ways.
- As a permanent rule:

```
f1[y_]:={y,y/2,y-3y^2};
f1[5] → {5, 5/2, -70}
```

- Or as a pure function (way 1):

```
f2=Function[{x,y}, x+y^2];
f2[2,3] → 11
```

- ... another way (way 2):

```
f3=(#1 + #2^2)&;
f3[2,3] → 11
```

- a function that returns a list of 2 rules:

```
f4[a_]:={2a->a^2, Rule[a,3a]}
```

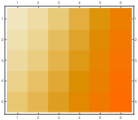
- f1: $\mathbb{R} \mapsto \mathbb{R}^3$, f2: $\mathbb{R}^2 \mapsto \mathbb{R}$, f3: $\mathbb{R}^2 \mapsto \mathbb{R}^1$.

- Examples of usage:
-

```
f1 /@ Range[3]
→ {{1,1/2,-2},{2,1,-10},{3,3/2,-24}},
```

-

```
Outer[f2, Range[0, 1, 0.25], Range[1,
2, 0.2]] // MatrixPlot
```

- → 

```
3
Transpose@{RandomReal[{0, 1}, %],
RandomVariate[NormalDistribution[],%]}
Apply[f3, %, {1}]
```

-

```
Table[f4[n], {n, 1, 2}]
→ {{2->1, 1->3}, {4->4, 2->6}}
```

Pure Functions + Simple Functions

- Define a function that represents mapping `x↦{x,Cos[x]}`. Implement 3 versions of the function: a) `g1` as a permanenut rule, b) `g2` as a pure function using `Function` command, and c) `g3` as a pure function *without* `Function` command.

```
g1 ...
g2 ...
g3 ...
{g1[b], g2[b], g3[b]}
```

- Define a vector/list `v = {3,6,7}`. Write an expression that uses `v` and `g1` (or `g2`, or `g3`) that produces the list `{{3,Cos[3]},{6,Cos[6]}, {7,Cos[7]}}`.

```
...
```

- Define a function `f5` that takes a vector/list of numbers (v) and returns an expression $\max(v) - \min(v)$. *E.g.* `f5 /@ {{3,4},{4,5},{1,3,4},{-3,8,2}}` should return `{1,1,3,11}`.

```
f5 ...
f5 /@ {{3, 4}, {4, 5}, {1, 3, 4}, {-3, 8, 2}}
```

Exercise

Pure Functions + Simple Functions (2)

- Assume you have an expression in `x`, *e.g.* `x^2`. Compute its derivative (use `D`) and evaluate the derivative in $x_0 = 5$ (use rules and `ReplaceAll`, resp. `/.`).

```
Clear[x]; x^2
...
```

- Define a function `tangent` that takes two arguments: 1) an expression `exprFx` in `x` variable, 2) a number `x0`. The `tangent` should return the tangent line (in `x0`) to plot of `exprFx`. Hints: use `D`, $f_{\text{tangent}}(x) = f'(x_0)(x - x_0) + f(x_0)$.

```
tangent[exprFx_, x0_] ...
x^2 (* define an expression in x *)
{%, tangent[%, 1], tangent[%, 2]}//Expand (* {func, tang. at x=1, tang. at x=2}*)
```

→ `{x^2, -1+2x, -4+4x}`

- Define a function `tangent2` that mathematically does the same as `tangent`, but the first argument is a function. Hint: use `Derivative`.

```
tangent2[func_, x0_] ...
f1[t_] := t^2 (* define a function *)
{f1[x], tangent2[f1, 1], tangent2[f1, 2]} // Expand
```

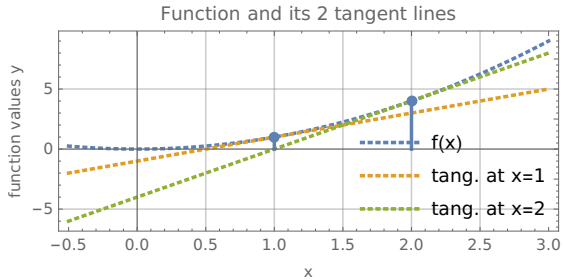
→ `{x^2, -1+2x, -4+4x}`

Exercise

Pure Functions + Simple Functions (2) – Plots

Plot the computed list of expressions in **x** from the previous slide.

```
... (* solution from the previous slide *)
{%, tangent[%, 1], tangent[%, 2]}//Expand (*or {f1[x],tangent2[f1,1],tangent2[f1,2]}
    //Expand*)
plt = Plot[%, {x, -0.5, 3}, GridLines -> Automatic, Frame -> True,
 FrameLabel ->{"x","function values y"}, PlotLegends ->Placed[{"f(x)", "tang. at
    x=1", "tang. at x=2"}, {Right, Bottom}], PlotStyle -> {{Thickness[0.007],
    Dotted}}, PlotLabel ->"Function and its 2 tangent lines", AspectRatio ->2/5];
listPlt = ListPlot[{{1, 1}, {2, 4}}, PlotStyle ->{{PointSize[Large]}}, Filling ->
    Axis, FillingStyle ->{Thickness[0.007]}];
Show[plt, listPlt]
```



Questions?

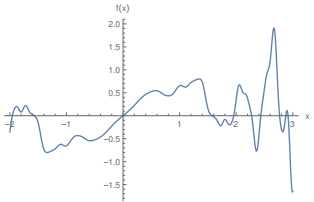
A8B17CAS
lukacjo1@fel.cvut.cz

December 2
Winter semester 2024/25

Voluntary homework

- Read file `"rec03_people.csv"`. The file contains records about people: Name, height [cm], weight [kg], age [years]. Determine 1) mean height, 2) minimal weight, 3) maximal age, 4) BMI[2] (body mass index) of people.
- Assume you have a list of expressions involving variable x: $\{\sin(x)\cos(x), x\sin^2(2x)\cos^2(2x), x^2\sin^3(3x)\cos^3(3x), x^3\sin^4(4x)\cos^4(4x), x^4\sin^5(5x)\cos^5(5x)\}$. Sum the elements and find the 2 roots between $x = 1$ and $x = 2$. I.e. find root of the function $f(x) = \sum_{n=1}^{5} x^{n-1}\sin^n(nx)\cos^n(nx)$ in the interval $x \in (1, 2)$ (hint: use **`FindRoot`**).

```
Table[x^(n - 1) Sin[x n]^n Cos[x n]^n, {n, 1, 5}]
```



[2] https://en.wikipedia.org/wiki/Body_mass_index

Voluntary homework (2)

- Read an RGB and a B/W image. RGB image is a matrix of triplets (`{r,g,b}`) – a 3D list, B/W image is a simple matrix.
  1. Define a function **`getRed`**: `{r,g,b}`↦ `{r,0,0}` – that returns just red part. Use the function with the RGB image.
  2. Define a function **`getGreen`**: `{r,g,b}`↦ `{g,0,0}` – that return just green part, but in place of the red. Use the function with the RGB image.
  3. Take B/W image and set a rectangle (100–150)x(120–170) to white (`1`)

```
SetDirectory[NotebookDirectory[] <> "data"]
FileNames[];
rgbPic = Import["im02_beans.tiff", "Data"]/255`; (* read RGB image *)
bwPic = Import["im03_desk.tiff", "Data"]/255`; (* read B/W image *)
getRed ...
getGreen ...
rgbPicRed = ...
rgbPicGreen = ...
bwPicModified = bwPic;
bwPicModified ...
Image /@ {rgbPic, rgbPicRed, rgbPicGreen, bwPic, bwPicModified}
```

Out[•]= 

Voluntary homework (3)

- Implement your-own simplified command **NIntegrate**. Assume an expression **expr** in variable **x**. Implement the "left rectangle" method: $\int_a^b f(x)\,\mathrm{d}x \approx S = h\sum_k f(x_k)$, $x_k = a + k\,h$, $x_k \le b$.
  1. Create list of **x**-values. Use **Range**.
  2. Evaluate **expr** in **x**-values. Use **ReplaceAll** (resp. **/.**) and a substitution rule for **x**.
  3. Sum the resulted $f(x_i)$ values and multiply by the step h

```
nIntegrate[expr_,a_,b_,h_] ...
Clear[x];
{nIntegrate[x^2, 0, 1, 1.0*^-3], NIntegrate[x^2, {x, 0, 1}]}
```

→ `{0.333833, 0.333333}`