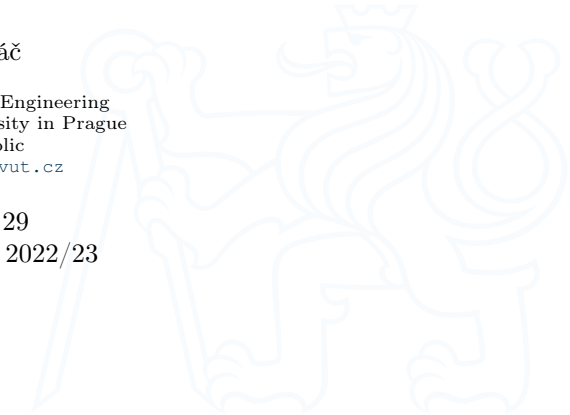
Lecture 11: Expressions in general.

A8B17CAS

Jozef Lukáč

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

November 29
Winter semester 2022/23



Outline

1. General Form of any Expression.
2. Types of Tokens in MATHEMATICA.
3. Ways to Input Expressions.
4. Forms of Expressions.
5. Input of Special Characters.
6. Properties of Expressions, Indexations.
   6.1 Simple Rules.
7. Apply and Map.
8. Pure Functions + Simple Functions.

(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	`+`, `->`, $\neq$
input to be ignored	`(* comment *)`

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
3^(1/4) + 1//N
{a,b,c}~Join~{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

Ways to Input Expressions.

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

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
```

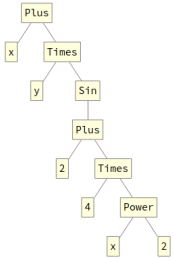
Input of Special Characters.

- 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 $\sqrt{4}$.

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];
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.
  - Eg. `expr2[[1]]`
    `expr2[[2]][[2]]`, resp. `expr2[[2,2]]`
    `expr2[[2,2,0]]`
    `mat={{-1,1,0},{5,3,1}};mat[[2,All]]`
    `mat[[;;,2]]`
    `mat[[;;,1;;2;;3]]` `mat[[3,{6,7}]]`

Properties of Expressions, Indexations.

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

```
SetDirectory[NotebookDirectory[] <> "data"];
fNames = FileNames[]
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 givem matrix, extract a submatrix such that it contains 4-th, 5-th row and 1-st, 3-rd and 5-th column of the original matrix.

Exercise

Properties of Expressions, Indexations.

Exercise

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

```
SetDirectory[NotebookDirectory[] <> "data"];
fNames = FileNames[]
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.

```
tab01[[;; , 2]] // Mean
tab01[[All , 3]] // Max
```

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

```
tab02 = Import["rec02_numbers.txt", "Data"];
tab02ex = tab02[[{4, 5}, {1, 3, 5}]];
```

Simple Rules.

- Rule says MATHEMATICA to take some part of expression and *substitute* it for another one (see `Rule` and `ReplaceAll`, resp. posfix form `/.`).
- *E.g.*
```
expr = a + b^2 + a c;
rule = a -> 3;
ReplaceAll[expr, rule]
```
- We can apply multiple rules, *e.g.*
```
rule2 = b->-5;
ReplaceAll[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 list of (nested) rules.
- *E.g.*
```
Clear[x]; sols = Solve[x^3 + x + 1 == 0, x] // N
x /. 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.
- Solve the system of equations

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

for x, y and extract values of $\{x, y\}$ from the solution-rules.

Exercise

Simple Rules.

- Solve the equation $x^2 + 4x - 1 = 0$ for x, use `Solve`, or `NSolve`. Extract values of x from the solution-rules.

```
Clear[x]; sols = Solve[x^2 + 4 x - 1 == 0, x]
x/.sols
```

- Solve the system of equations

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

for x, y and extract values of $\{x, y\}$ from the solution-rules.

```
Clear[x, y];
sols = NSolve[{x^2 + x y - 1 == 0, y^2 + x + 2 == 0}, {x, y}]
{x,y}/.sols
```

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 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}]
```

- `Apply` takes a head (*e.g.* `Plus`, `List`, `f3`, ...) and *substitutes* head of the input expression, *e.g.* `Apply[f3,myList]`
- Also, we can specify the *"application level"*, *e.g.* `Apply[f3, myList, {2}]`

Apply and Map.

- Some examples:

```
v = {5, 4, 6};
Apply[Times, v]
Plus @@ v
```

- Use `Apply` to get the sum of rows in the following matrix:

```
mat={{0,1},{5,4}};
```

Exercise

Apply and Map.

- Some examples:

```
v = {5, 4, 6};
Apply[Times, v]
Plus @@ v
```

- Use `Apply` to get the sum of rows in the following matrix:

```
mat={{0,1},{5,4}};
TreeForm @ mat
Apply[Plus, v,{1}]
```

Exercise

Pure Functions + Simple Functions

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

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

- Or as a pure function (way 1):

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

- ... another way (way 2):

```
f3=(#1 + #2^2)&;
f3[2,3]
```

Pure Functions + Simple Functions

- Declare a function `g` that *takes one argument*: `a`, and returns a *list*: `{a,Cos[a]}`.
- Declare vector `v`, `v={3,6,7};`. Use `Map` command and function `g` to create vector `gv = {{3,Cos[3]},{6,Cos[6]}, {7,Cos[7]}}`.

Exercise

Pure Functions + Simple Functions

- Declare a function `g` that *takes one argument*: `a`, and returns a *list*: `{a,Cos[a]}`.
  `g={#,Cos[#]}&;`
- Declare vector `v`, `v={3,6,7};`. Use `Map` command and function `g` to create vector `gv = {{3,Cos[3]},{6,Cos[6]}, {7,Cos[7]}}`.

Exercise

Pure Functions + Simple Functions

- Declare a function `g` that *takes one argument*: `a`, and returns a *list*: `{a,Cos[a]}`.
  `g={#,Cos[#]}&;`
- Declare vector `v`, `v={3,6,7};`. Use `Map` command and function `g` to create vector `gv = {{3,Cos[3]},{6,Cos[6]}, {7,Cos[7]}}`.
  `g /@ v`

Exercise

Questions?

A8B17CAS
lukacjo1@fel.cvut.cz

November 29
Winter semester 2022/23