# Lecture 12: Lists, Rules and Patterns.

A8B17CAS

Jozef Lukáč

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

December 6
Winter semester 2022/23

# Outline

1. Creation of matrices and vectors.
2. Accessing parts of matrices/vectors/expressions.
3. Rearranging lists.
4. Max, MaximalBy, DeleteDuplicates.
5. Lists as Sets.
6. Functions for testing properties of numbers.
7. Patterns
8. Functions for testing structural properties of expressions.
9. Set vs. SetDelayed, Rule vs. RuleDelayed
10. Putting constraints on patterns and transformation rules.
11. Cases, Count, Position, Select

# Creation of matrices and vectors.

Common commands to create matrix and vector:

- `Range`, `Table`, (`Array`), `IdentityMatrix`, `DiagonalMatrix`, `Subdivide`, *e.g.*
  `{Range[5], Range[-3,1], Range[2,13,3]}`
  → `{{1, 2, 3, 4, 5}, {-3, -2, -1, 0, 1}, {2, 5, 8, 11}}`
  in MATLAB it would be `1:5, -3:1, 2:3:13`.
- Use function `f[n]`, `f[m,n]`, `f[m,n,o]`, . . . to specify each element of a vector/matrix/multidimensional array, *e.g.*:
  `Table[f[n],{n,5}]` → `{f[1], f[2], f[3], f[4], f[5]}`
  `Table[i^2 Sqrt[j],{i,2,4},{j,3, 8, 2}]` → {{4$\sqrt{3}$, 4 $\sqrt{5}$, 4 $\sqrt{7}$}, {9 $\sqrt{3}$, 9 $\sqrt{5}$, 9 $\sqrt{7}$}, {16 $\sqrt{3}$, 16 $\sqrt{5}$, 16 $\sqrt{7}$}}
- `Table[0,{n1Max},n2Max,...]` is equivalent to `zeros(n1Max,n2Max,...)` in MATLAB.
- `Table[1,{n1,n1Max},{n2Max},...]` is equivalent to `ones(n1Max,n2Max,...)` in MATLAB.

# Creation of matrices and vectors.

- `Table[RandomReal[],n1Max,{n2Max},...]`, resp. `RandomReal[{0,1},{n1Max,n2Max,...}]` is equivalent to `rand(n1Max,n2Max,...)` in MATLAB.
- For generation of random number from general distribution, use `RandomVariate`, *e.g.* `RandomVariate[NormalDistribution[],{n1Max,n2Max,...}]` is equivalent to `randn(n1Max,n2Max,...)` in MATLAB.
- `IdentityMatrix[n]` is equivalent to `eye(n)` in MATLAB.
- To create a diagonal matrix from a vector, use `DiagonalMatrix[vec]` (equivalent to `diag(vec)` in MATLAB).
- To extract a diagonal vector from a matrix, use `Diagonal[mat,i]` (equivalent to `diag(mat,i)` in MATLAB).
- Equivalent to MATLAB `linspace(a,b,n)` is `Subdivide[a,b,n-1]`.
- To concatenate matrices, use `Join`, *e.g.*
```
mat1 = {{1, 2, 3}, {4, 5, 6}}; mat2 = {{7, 8, 9}, {10, 11, 12}};
MatrixForm /@ {mat1, mat2, mat1~Join~mat2, Join[mat1, mat2, 1],
Join[mat1, mat2, 2]}
```
- Get dimensions of matrix/expression by `Dimensions`, *e.g.* `Dimensions[mat1]`.

# Accessing parts of matrices/vectors/expressions.

We can access different parts of matrices/expressions by the following commands:

- `Part[expr,idc]`, resp. `expr[[idc]]` – by specifying the index of a part, *e.g.*
  `Part[mat1,2, {2, 3}],  (3 x + Sin[x^3])[[2, 1, {1, 2}]]`
- `Take[expr,n]` – Get the first `n` elements from the expression/list. *e.g.*
  `Take[a^8 b c^4 d,2]` → `a^8 b`
- `Drop[list,n]` – Returns list with its first `n` elements dropped, *e.g.*
  `Drop[{4, 5, 9, -2}, 2]` → `{9, -2}`
- `Most[expr]` – Returns expression with its last element removed, *e.g.*
  `Most[9 + x^2 + Sin[x]]` → `9+x^2`
- `Last[expr]` – Returns the last part of expression, *e.g.*
  `Last[f[a^2, b, c^4, d, e]]` → `e`
- `First[expr]` – Gives the first part of expression, *e.g.*
  `First[a^8 b c]` → `a^8`
- `Rest[expr]` – Returns the expression with the first element removed, *e.g.*
  `Rest[a^8 b c]` → `b c`

# Create matrix, example

- Create the following matrix

$$\begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 5 & 0 & 0 & r_1 & r_2 \\ 0 & 6 & 0 & r_3 & r_4 \\ 0 & 0 & 7 & r_5 & r_6 \end{bmatrix}.$$

where the $r_i$ are random number from the uniform distribution on $(0, 1)$.

Exercise

# Create matrix, example

- Create the following matrix

$$\begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 5 & 0 & 0 & r_1 & r_2 \\ 0 & 6 & 0 & r_3 & r_4 \\ 0 & 0 & 7 & r_5 & r_6 \end{bmatrix}.$$

where the $r_i$ are random number from the uniform distribution on $(0, 1)$.

```
myJoin[vec1_, vec2_] := Join[vec1, vec2, 2];
{Range[-2, 2]}~Join~(Table[0, 3, 2]~myJoin~Table[1, 3, 3])~
Join~(DiagonalMatrix[{5, 6, 7}]~myJoin~ Table[RandomReal[], 3,
2]) // MatrixForm
```

Exercise

# Rearranging lists.

Some useful commands to rearrange lists/expressions.

- `Sort[list,orderingFuncion]` – sorts list according the ordering function (takes 2 elements and returns True/False).
- `SortBy[list,f]` – sorts list in the order defined by applying `f` to each element.
- `RotateLeft[expr,n]` – cycles the elements in expr `n` positions to the left.
- `RotateRight[expr,n]` – cycles the elements in expr `n` positions to the right.
- `Transpose[list]` – transposes the first two levels in list.
- Examples,

```
Sort[f[1,3,2]] → f[1,2,3]
SortBy[{{a,1},{{3,1},3},{x^2,2}},Last] →
{{a,1},{x^2,2},{{3,1},3}}
RotateRight[{5,-1,4,2},2] → {4,2,5,-1}
```

# Max, MaximalBy, DeleteDuplicates.

- `Max[list]` – returns maximal number from the list of numbers.
- `MaximalBy[list,f]` – returns a list of elements for which `f[e`$_i$`]` is maximal.
- `DeleteDuplicates[list]` – deletes all duplicates from list.
  `Sort@DeleteDuplicates[list]` and `Union[Sequence @@ {#} & /@ list]` are equivalent to `unique(list)` in MATLAB.
- Examples,

```
Max[{5,4,6,-2,3}] → 6
MaximalBy[{{a,1},{{3,1},3},{x^2,2}},Last] → {{3,1},3}
DeleteDuplicates[{2,3,-1,2,5,6,3,2,8}] → {2,3,-1,5,6,8}
```

# Max, MaximalBy, DeleteDuplicates.

- From the folder "data" read "rec01_containers.txt". Sort the records according to weight of empty container (the third column), and according to the volume of water in it (the second column).

- BMI index (Body mass index) is defined as weight[kg]/height$^2$[m$^2$]. Read "rec03_people.csv" and get the record and name of the person with maximal BMI. Record contains: name, height [cm], weight [kg], age [years].

Exercise

# Max, MaximalBy, DeleteDuplicates.

- From the folder "data" read "rec01_containers.txt". Sort the records according to weight of empty container (the third column), and according to the volume of water in it (the second column).
```
tab01 // TableForm
SortBy[tab01, Last] // TableForm
SortBy[tab01, Part[#, 2] &] // TableForm
```
- BMI index (Body mass index) is defined as weight[kg]/height$^2$[m$^2$]. Read "rec03_people.csv" and get the record and name of the person with maximal BMI. Record contains: name, height [cm], weight [kg], age [years].
```
f1[rec_] := rec[[3]]/(rec[[2]]/100)^2;
f2[{name_, h_, w_, a_}] := w/(h/100)^2;
MaximalBy[tab02, f2]
Part[#, 1] & /@ MaximalBy[tab02, f2]
```

Exercise

# Lists as Sets.

To work with lists as sets, following commands are useful.

- `Union[list1,list2,...]` – gets a union of the lists.
- `Intersection[list1,list2,...]` – gets the intersection of all the lists.
- `Complement[allElems,list1,list2,...]` – gives element that are not present in any list.
- Examples

```
Union[f[6,3,4],f[1,3]] → f[1,3,4,6]
Intersection[{5, 3, 6, 1, 3}, {3, 6, 4}] → {3,6}
Complement[Range[6], {5, 3}, {2, 3}] → {1,4,6}
```

# Functions for testing properties of numbers.

Test functions in MATHEMATICA usually end with `Q` (query, question) command/function. And return `True` or `False`.

- `IntegerQ` – whether the number is integer.
- `EvenQ` – whether the number is even.
- `PrimeQ` – whether the number is prime.
- `VectorQ` – whether the input is a simple list (without nested lists).
- `MatrixQ` – whether the input is a matrix – list of lists (of same length).
- `NumericQ` – whether the input object is a numeric object.
- Examples:

```
IntegerQ /@ {1,2/3,Sqrt[3],-4}
MatrixQ /@ {5, {1,2},x^2, {{1,2},{3,4}}}
```

# Patterns

FUNDAMENTAL PRINCIPLE of MATHEMATICA: *Take any expression, and apply transformation rules until the result no longer changes.*

- *Pattern is representation of a group/class of expressions*, *e.g.* `f[x_]` means `f[anyExpression]`.
- Examples of patterns:

| | |
|---|---|
| `_` | any single expression, |
| `x_` | any single expression to be named `x`, |
| `__` | any *sequence* of one or more expressions, |
| `x__h` | sequence of expressions, all of whose heads are `h`, |
| `___` | any sequence of zero or more expressions, |
| `x___` | any sequence of zero or more expressions named `x`, |
| `x___h` | sequence of zero or more expressions, all of whose heads are `h`, |
| `f[n_]` | `f` with any argument, named `n`, |
| `f[n_,m_]` | `f` with two arguments, named `n` and `m`, |
| `x^n_` | `x` to any power, with the power named `n`, |
| `x_^n_` | any expression to any power |

# Patterns

| | |
|---|---|
| `a_ + b_` | a sum of two expressions, |
| `{a1_,a2_}` | a list of two expressions |

- patterns for objects with specified Heads:

| | |
|---|---|
| `x_h` | an expressions with head `h`, |
| `x_Integer` | an expressions with head `Integer`, |
| `x_Real` | an expressions with head real number, |
| `x_Complex` | a complex number, |
| `x_List` | a list, |
| `x_Symbol` | a symbol |

# Functions for testing structural properties of expressions.

- `Equal`, resp. `==` – true if both sides are identical.
- `OrderedQ[list]` – checks whether the list is sorted.
- `MemberQ[list,form]` – checks whether an element from a list matches the form.
- `FreeQ[expr,form]` – checks whether no subexpression matches the form.
- `MatchQ[expr,form]` – true if the pattern form matches expr.
- `ValueQ[expr]` – whether a value has been defined for expr.
- `AtomQ[expr]` – true if the expr is atomic expression (cannot be divided into subexpression).
- Examples:

```
OrderedQ[{1,x^3,x^4}] → True
expr = Sin[x^3] + x y; MemberQ[expr, y, {2}] → True
{FreeQ[expr, _^3], FreeQ[expr, _^4]} → {False, True}
```

# Set vs. SetDelayed, Rule vs. RuleDelayed

- Note the difference in the assignments:

```
f1=Random[]; (*Set[f1,Random[]]*)
f2:=Random[]; (*SetDelayed[f2,Random[]]*)
{f1,f1,f2,f2}
```

- ...and in the rules (`rule` – `Rule`, `rule2` – `RuleDelayed`):

```
Clear[p]; a = 5; rule = a_^3 -> a; Table[p^i, {i, 4}] /.  rule
Clear[p, a]; rule2 = a_^3 :> a; Table[p^i, {i, 4}] /.  rule2
{rule, rule2}
```

- `SetDelayed` and `RuleDelayed` evaluate when called/used, whereas ordinary `Set` and `Rule` evaluate when defined.

# Putting constraints on patterns and transformation rules.

We can put a constraint on a pattern, so that it matches only if the condition is applied.

- one way by `?boolFunction`, *e.g.*
  `Clear[ff,x]; ff[x_?EvenQ]:=x/2; ff/@{1,2}`
  note, that the following will not work
  `Clear[ff,x]; ff[x_?EvenQ[x]]:=x/2; ff/@{1,2}`
- another way by `/;resultOfABoolFuncion`, *e.g.*
  `Clear[gg,x]; gg[x_/;EvenQ[x]]:=x/2; gg/@{3,4}`
  note, that the following will not work
  `Clear[gg,x]; gg[x_/;EvenQ]:=x/2; gg/@{3,4}`
- By the second way, we can constrain also rules and definitions, *e.g.*
  `rule=Times[a_,b_]:>a Sin[b]/;NumericQ[a]; x+2z^3 + x z/.rule` → `x + x z + 2 Sin[z^3]`
  `Clear[f2, f3]; f2[x_/;x^2>5]:=x^3; f3[x_]:=x^3/;x^2>5; {f2[2],f2[3],f3[2],f3[3]}`

# Putting constraints on patterns and transformation rules.

- Consider vector `vec=Table[RandomInteger[{-3,3}],10];`. Define and apply a `rule` that takes any negative number in `vec` and transforms it to its square, *i.e.* if $x < 0$ return $x^2$ otherwise do nothing.
  `rule=`
- Consider a list of vectors:
  `lst={Range[4],RandomInteger[{0,4},3],Table[0,5], {1,3}};`
  Define and apply a `rule2` that takes a list of numbers and if the number of elements in the list is more than 3 transforms the list by taking just first three elements from it.
  `rule2=`
  `Replace[lst,rule2,1]`

Exercise

# Putting constraints on patterns and transformation rules.

- Consider vector `vec=Table[RandomInteger[{-3,3}],10];`. Define and apply a `rule` that takes any negative number in `vec` and transforms it to its square, *i.e.* if $x < 0$ return $x^2$ otherwise do nothing.
```
rule=x_/;x<0:>x^2; vec/.rule
```
- Consider a list of vectors:
```
lst={Range[4],RandomInteger[{0,4},3],Table[0,5], {1,3}};
```
Define and apply a `rule2` that takes a list of numbers and if the number of elements in the list is more than 3 transforms the list by taking just first three elements from it.
```
rule2=
Replace[lst,rule2,1]
```

Exercise

# Putting constraints on patterns and transformation rules.

- Consider vector `vec=Table[RandomInteger[{-3,3}],10];`. Define and apply a `rule` that takes any negative number in `vec` and transforms it to its square, *i.e.* if $x < 0$ return $x^2$ otherwise do nothing.
  `rule=x_/;x<0:>x^2; vec/.rule`
- Consider a list of vectors:
  `lst={Range[4],RandomInteger[{0,4},3],Table[0,5], {1,3}};`
  Define and apply a `rule2` that takes a list of numbers and if the number of elements in the list is more than 3 transforms the list by taking just first three elements from it.
  `rule2=x_List/;Length[x]>3:>Take[x,3];`
  `Replace[lst,rule2,1]`

Exercise

# Cases, Count, Position, Select

- `Cases[expr,form]` – gives a list of the elements that match the pattern,
- `Cases[expr,pattern->rhs]` – gives a list of the values or `rhs` corresponding to the elements that match the pattern.
- `Count[expr,pattern]` – gives the number of elements in expr that match the pattern.
- `Position[expr,pattern]` – give a list of postions at which objects matching pattern appear in expr.
- `Select[list, crit]` – returns all elements for which crit[$e_i$] evaluates as `True`.
- Examples:
  `Cases[{{1, 2}, {2}, {3, 4, 1}, {5, 4}, {3, 3}}, {_, _}]` → `{{1, 2}, {5, 4}, {3, 3}}`
  `Count[5x^5+3 y^5,5,Infinity]` → `3`
  `Position[{1 + x^2, 5, x^4, a + (1 + x^2)^2}, x^_, 2]` → `{{1, 2}, {3}}`
  `Select[Range[10],IntegerQ[Sqrt[#]]&]` → `{1,4,9}`

# Cases, Count, Position, Select

- Consider matrix `mat`,

```
Needs["Combinatorica`"]; nMax = 5;
mat = Transpose@{RandomInteger[{0, 10}, nMax],
Combinatorica`RandomPermutation[nMax]}
```

Choose elements from the matrix `mat` such that *first element* − *second element* $< 3$. Do it twice, once with `Cases` and then with `Select`.

Exercise

# Cases, Count, Position, Select

- Consider matrix `mat`,

```
 Needs["Combinatorica`"]; nMax = 5;
 mat = Transpose@{RandomInteger[{0, 10}, nMax],
Combinatorica`RandomPermutation[nMax]}
```

Choose elements from the matrix `mat` such that *first element* − *second element* $< 3$. Do it twice, once with `Cases` and then with `Select`.

```
 Cases[mat, {n1_, n2_} /; n1 - n2 < 3]
 Select[mat, Part[#, 1] - Part[#, 2] < 3 &]
```

Exercise

# Questions?

A8B17CAS
lukacjo1@fel.cvut.cz

December 6
Winter semester 2022/23