LaTeX:
from dummy to TeXnician
Command creation. How TeX works-2

Anton Lioznov

Skoltech,
Project Center of Omics Technologies and Advanced Mass Spectrometry

ISP 2025,
lesson 5

What we will know?

Technical agreements

Simple command creation and style files creation

Programming

Debugging

What we will know?

- Technical agreements
- Simple command creation and style files creation
- Programming
- Debugging

Agreements

inclass/outclass versions

- two slightly different versions for class and home
- class version is more interactive and contains less information
- ⟷ this line will be shown only at home version

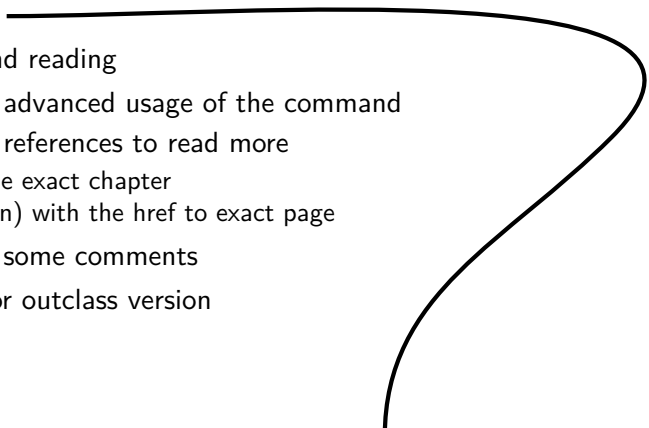
Frame for home

Agreements

II

Footnotes

- For second reading
- Contains advanced usage of the command
- Contains references to read more
  - to the exact chapter
  - (often) with the href to exact page
- Contains some comments
- Mostly for outclass version

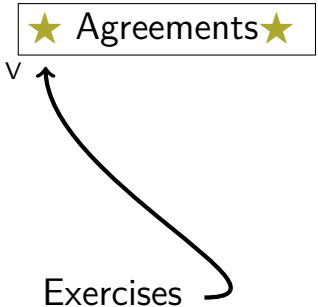


Like this

Agreements
III

Addition information – “magic”

- To have the full picture
- Not to analyze or to puzzle out in class



- To work in class

Special thanks to

Our TAs:

- Peter Borisovets
- Pavel Kuzmin
- Anna Litvin

What we will know?

Technical agreements

Simple command creation and style files creation

Programming

Debugging

Why we are studying it?

What with command creation?

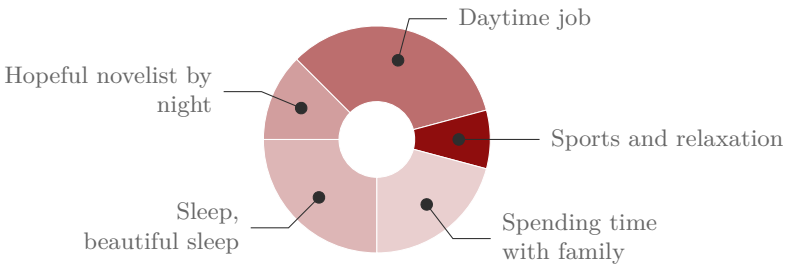
For most documents you need no command creation knowledge: just using the existing.

But commands creation skill will allow you:

- Dramatically shorten the time and increase the pleasure of the process
- Kill the routine
- Create useful thing to share with others
- Understand and be able to change the code from templates
- Usually to create a simpler UI you need a more difficult backend

To create a clear syntax for yourselves

```
\wheelchart{1.5cm}{0.5cm}{%
  6/8em/accent!30/{Sleep,\\beautiful sleep},
  3/8em/accent!40/Hopeful novelist by night,
  8/8em/accent!60/Daytime job,
  2/10em/accent/Sports and relaxation,
  5/6em/accent!20/Spending time with family
}
```



(From altaCV template)

What we will know?

Simple command creation and style files creation
- Command creation
- Style and class files

Create new command

without arguments

```
\newcommand{\lookAtMe}{\vbox{I'm mister Meeseeks look at me!}}
\newcommand{\dfdx}{\ensuremath{\frac{\partial f}{\partial x}}}
\begin{document}
    \lookAtMe \lookAtMe \lookAtMe \lookAtMe
    \dfdx
    $$\dfdx = 5x$$
\end{document}
```

Create new command

without arguments

`\newcommand{<\commandname>}{<code>}` to create new macros

```
\newcommand{\lookAtMe}{\vbox{I'm mister
↪  Meeseeks look at me!}}
\newcommand{\dfdx}{\ensuremath{\frac{\partial
↪  f}{\partial x}}}
\begin{document}
    \lookAtMe \lookAtMe \lookAtMe
    ↪  \lookAtMe
    \dfdx
    $$\dfdx = 5x$$
\end{document}
```

I'm mister Meeseeks look at me!
I'm mister Meeseeks look at me!
I'm mister Meeseeks look at me!
I'm mister Meeseeks look at me!
$\frac{\partial f}{\partial x}$

$$\frac{\partial f}{\partial x} = 5x$$

Recreate command

`\renewcommand` to recreate already created command

```
\newcommand{\lookAtMeS}{\vbox{I'm mister
↪  Meeseeks look at me!}}
\renewcommand{\lookAtMeS}{\vbox{I'm missis
↪  Meeseeks look at me!}}

\newcommand{\dfdx}{\ensuremath{\frac{\partial
↪  f}{\partial x}}}

\begin{document}
    \lookAtMeS \lookAtMeS \lookAtMeS
    ↪  \lookAtMeS
    \dfdx
    $$\dfdx = 5x$$
\end{document}
```

I'm missis Meeseeks look at me!
I'm missis Meeseeks look at me!
I'm missis Meeseeks look at me!
I'm missis Meeseeks look at me!
$\frac{\partial f}{\partial x}$

$$\frac{\partial f}{\partial x} = 5x$$

Create new command

with arguments

```
\newcommand{\lookAtMe}[1]{\vbox{I'm mister #1 look at me!}}
\newcommand{\dfdx}[2]{\ensuremath{\frac{\partial #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha} \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

Create new command

with arguments

\newcommand{<commandname>}[<number of args]{<code>}. Refer to arg as #1, #2, ...

```
\newcommand{\lookAtMe}[1]{\vbox{I'm mister
↪  #1 look at me!}}
\newcommand{\dfdx}[2]{\ensuremath{\frac{\partial
↪  #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha}
    ↪  \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

I'm mister Gosha look at me!
I'm mister Misha look at me!
I'm mister Tema look at me!

$$\frac{\partial g}{\partial y}$$

$$\frac{\partial v}{\partial z} = 5x$$

Define macros

In TEX you can define new macros via \def.

```
\def\lookAtMe#1{\vbox{I'm mister #1 look at me!}}
\def\dfdx#1#2{\ensuremath{\frac{\partial #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha} \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

Define macros

In TEX you can define new macros via \def.

```
\def\lookAtMe#1{\vbox{I'm mister #1 look
↪  at me!}}
\def\dfdx#1#2{\ensuremath{\frac{\partial
↪  #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha}
    ↪  \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

I'm mister Gosha look at me!
I'm mister Misha look at me!
I'm mister Tema look at me!

$$\frac{\partial g}{\partial y}$$

$$\frac{\partial v}{\partial z} = 5x$$

Use \global prefix to define macros not just inside "group".

Use \long prefix to define macros that can have multiple paragraphs as an argument.

Compare LaTeX and TeX

```
\newcommand{\lookAtMe}[1]{\vbox{I'm mister #1 look at me!}}
\newcommand{\dfdx}[2]{\ensuremath{\frac{\partial #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha} \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

```
\def\lookAtMe#1{\vbox{I'm mister #1 look at me!}}
\def\dfdx#1#2{\ensuremath{\frac{\partial #1}{\partial #2}}}
\begin{document}
    \lookAtMe{Gosha} \lookAtMe{Misha} \lookAtMe{Tema}
    \dfdx{g}{y}
    $$\dfdx{v}{z} = 5x$$
\end{document}
```

{}

```
\newcommand{\lookAtMe}[1]{\vbox{I'm mister
↪  #1 look at me!}}
\lookAtMe Anton

\lookAtMe\LaTeX
```

I'm mister A look at me!
nton
I'm mister LaTeX look at me!

⟷ The braces just indicates atomicy!

{}

```
\newcommand{\lookAtMe}[1]{\vbox{I'm mister
↪  #1 look at me!}}

\lookAtMe Anton

\lookAtMe\LaTeX

$$x^y$$

$$x^\sigma $$

$$x^yz$$

$$x^{yz}$$
```

I'm mister A look at me!
nton
I'm mister LaTeX look at me!

$$x^y$$

$$x^\sigma$$

$$x^y z$$

$$x^{yz}$$

$\longleftrightarrow$ The braces just indicates atomicy!

what is “atomic” in TEX

- Commands `\somecommand`
- Symbols
- Everything in braces {}

Command creation inside command creation

As simple as `\newcommand{\name}{\newcommand{\othername}{smth}}`

1. In the inner command, you can refer to the argument of outer command as #1
2. In the inner command, you can refer to the argument of inner command as ##1

Sometimes you can see something like

```
\newcommand{\photo}[1]{\renewcommand{\photo}[#1]}
```

It provides the following usage: You can store something at first usage as `\photo{myface.png}` and then refer to it as just `\photo`

The scope

The braces at command definition and at command usage ommited. If you want your code to have local effect – provide an extra braces:
not

```
\newcommand{\htext}[1]{\Huge text}
```

but

```
\newcommand{\htext}[1]{{\Huge text}}
```

New environment

use \newenvironment{<name>}{<code at begin>}{<code at end>}
or \renewenvironment

New command with optional arguments

```
\newcommand{\testOpt}[2][my default opt
↪  arg]{
optional arg: #1\\
default arg: #2\\
}

\testOpt[non-def opt arg]{defarg}\\
\testOpt{defarg}
```

optional arg: non-def opt arg
default arg: defarg

optional arg: my default opt arg
default arg: defarg

Use syntax `\newcommand{\<cmdname>}[total_arg_num][defaults]{<code>}`

New command with optional arguments

using package

```
\usepackage{xargs}

\newcommandx{\testOpt}[3][3=def opt val]{
    arg 1: #1\\
    arg 2: #2\\
    arg 3: #3\\
}

\testOpt{arg 1}{arg 2}[arg 3]\\
\testOpt{arg 1}{arg 2}\\
```

arg 1: arg 1
arg 2: arg 2
arg 3: arg 3

arg 1: arg 1
arg 2: arg 2
arg 3: def opt val

Use \usepackage{xargs} and \newcommandx (notice **x** at the end)

New command with key=value syntax

keyval package

```
\usepackage{keyval}
\define@key{test}{color}{%
  \def\thiscolor{#1}}

\newcommand{\mycommand}[1]{%
  \def\thiscolor{}% default
  \setkeys{test}{#1}%
   the color is \thiscolor}

\mycommand{color=red}
```

the color is red

Use \usepackage{keyval} and \define@key, \setkeys

New command with key=value syntax

keyval package

```
\usepackage{pgfkeys}
\pgfkeys{my key/.code=The value is '#1'.,
↪  otherkey/.code=~ the \textbf{scnd}
↪  value is '#1'}
\pgfkeys{my key=hi!, otherkey=AA}
```

Use \usepackage{pgfkeys}

The value is 'hi!'. the **scnd** value is 'AA'

Where to put your own commands

1. You can put it into document preamble
2. You can put it inside document whenever you want. Then:
   - The command can be used only after it's definition
   - The command definition is LOCAL: the scope of the visibility is the GROUP
3. You can put it into style or class files

What we will know?

- Simple command creation and style files creation
  - Command creation
  - Style and class files

The first lines in .cls and .sty files

Class:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{<class-name>}[<date in YYYY/MM/DD> <other info>]
```

Style:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{<package-name>}[<date in YYYY/MM/DD> <other info>]
```


man: 3.3.1 https://www.latex-project.org/help/documentation/clsguide.pdf

Special syntax

You can use the same commands: `\newcommand` and `\usepackage` inside .sty and .cls files, but it is better to change them to:

`\newcommand`	$\rightarrow$	`\providecommand`
`\usepackage`	$\rightarrow$	`\RequirePackage`
`\documentclass`	$\rightarrow$	`\LoadClass`

Passing options

To use a syntax like `\documentclass[14pt]{beamer}` or `\usepackage[english]{babel}` you need to declare options in you .sty/.cls file.

examplepassing.sty:

```
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{examplepassing}[2018/01/16 the simple package with passing options]

\newcommand{\hello}{}
\DeclareOption{first}{
    \renewcommand{\hello}{Username}
}
\DeclareOption{second}{
    \renewcommand{\hello}{World}
}
\ProcessOptions
```

.tex files:

```
\usepackage[first]{examplepassing}
\begin{document}
    \hello
\end{document}
```

Username

```
\usepackage[second]{examplepassing}
\begin{document}
    \hello
\end{document}
```

World

Passing previously unknown options

Use \DeclareOption*{<code with \CurrentOption variable>} to process previously unknown options.

Useful to pass commands to class:

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{letter}}
\ProcessOptions\relax
\LoadClass[a4paper]{letter}
```

New package with key=value syntax

packkvexample.sty:

```
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{packkvexample}[2018/01/16 the simple keyval package]
\RequirePackage{kvoptions}
% process the arguments for the package
\SetupKeyvalOptions{
    family=KVAR,
    prefix=KVAR@
}
\DeclareStringOption[noarg]{myarg}[defaultarg]
\ProcessKeyvalOptions*

\newcommand{\showarg}{\KVAR@myarg}
```

.tex files:

```
\usepackage[myarg=hello?]{packkvexample}
\showarg

\usepackage[myarg]{packkvexample}
\showarg

\usepackage{packkvexample}
\showarg
```

hello?

defaultarg

noarg

Class or package?

- No “programming-level” restrictions
- The “logical-level” difference: If the commands could be used with any document class, then make them a package; and if not, then make them a class.

https://www.latex-project.org/help/documentation/clsguide.pdf: 2.3

Code conventions

- if command is for author, try short name and lowcase: \section, \emph and \times
- if command is for package and class creator, use CamelCase: \InputIfFileExists \RequirePackage \PassOptionsToClass
- There are the internal commands used in the LaTeX implementation, such as `\@tempcnta`, `\@ifnextchar` and `\@eha`: most of these commands contain `@` in their name, which means they cannot be used in documents, only in class and package files

If you wish to use command with @ in .tex, use `\makeatletter`, <use command>, `\makeatother`.

What we will know?

Technical agreements

Simple command creation and style files creation

Programming

Debugging

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

TEX is Turing-complete language

In basic words, it means, that you can write in TEX and LATEX any algoritms, that you can write in C++, Java, Python... Moreover, some TEXsyntax is really familiar to functional languages

https://stackoverflow.com/questions/2968411/ive-heard-that-latex-is-turing-complete-are-there-any-programs-written-in-late

6/Articles/LaTeX_is_More_Powerful_than_you_Think_-_Computing_the_Fibonacci_Numbers_and_Turing_Completeness

http://sdh33b.blogspot.com/2008/07/icfp-contest-2008.html

Reminder: Define macros

In TEX you can define new macros via \def.

```
\def\lookAtMe#1{\vbox{I'm mister #1 look
↪  at me!}}
\def\dfdx#1#2{\ensuremath{\frac{\partial
↪  #1}{\partial #2}}}

\lookAtMe{Gosha} \lookAtMe{Misha}
↪  \lookAtMe{Tema}
\dfdx{g}{y}
$$\dfdx{v}{z} = 5x$$
```

I'm mister Gosha look at me!
I'm mister Misha look at me!
I'm mister Tema look at me!
$\frac{\partial g}{\partial y}$

$$\frac{\partial v}{\partial z} = 5x$$

Use \global prefix to define macros not just inside "group".

Use \long prefix to define macros that can have multiple paragraphs as an argument.

Define with pattern matching

The syntax with writing each argument seems to be an over-use. But it is needed because of *pattern matching*

```
\def\parseLine#1, #2\par{arg1: #1\\ arg2:
↪  #2 }

\parseLine Hello, World
```

arg1: Hello
arg2: World

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

Compare strings (macros)

```
\def\ttest#1#2{
    \def\a{#1}
    \def\b{#2}
    \ifx\a\b
        yes
    \else
        no
    \fi
}

\ttest{ab}{ab} \ttest{ba}{ab}
```

yes no

\ifx\<first>\<second> <code1> [\else <code2>] \fi

Compare numbers

```
\def\ttest#1#2{
    \ifnum#1>#2 yes \else no \fi
}

\ttest{2}{1}
\ttest{1}{2}

\def\testCase#1{
    \ifcase#1 first
        \or second
        \or third
        \else don't know
    \fi
}
\testCase{2} \testCase{55}
```

yes no
third don't know

`\ifnum<first><operator><second> <code1> [\else <code2>] \fi`. Only "=", ">" or "<" are allowed.

Use `\ifcase` to check different stuff. Also use `\ifodd` to check if num is odd or even

Compare in LaTeX

```
\usepackage{xstring}
\def\ttest#1#2{
    \IfStrEq{#1}{#2}{yes}{no}
}

\ttest{ab}{ab} \ttest{ba}{ab}
```

yes no

\usepackage{xstring}

also see \usepackage{ifthen}

you can check if you are in XeLaTeX by \usepackage{ifxetex}

read about the xstring package! It has lots of things: strings, substrings,.. what to do with them,.. replace strings etc

/47576/combining-ifxetex-and-ifluatex-with-the-logical-or-operation

Check modes

```
\def\testm{\ifmmode yes \else no \fi}
\testm $\testm$

\def\testv{\ifvmode yes \else no \fi}
\testv \leavevmode \testv

\def\testh{\ifhmode yes \else no \fi}
\testh \leavevmode \testh

\def\testi{\ifinner yes \else no \fi}
$\testi$ $$\testi$$
```

no yes
yes no
no yes
yes

no

- `\ifmmode` to check if in mathematical mode
- `\ifvmode` to check if in vertical mode
- `\ifhmode` to check if in horizontal mode
- `\ifinner` to check if TEX is in internal vertical mode, or restricted horizontal mode, or (nondisplay) mathmode

other "if"'s can be found in kn: 20

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

Loop

```
\newcount\icount
\icount=10
\loop A?
    \ifnum\icount>0
        B! \advance \icount by -1
        \repeat
```

A? B! A? B! A? B! A? B!
A? B! A? B! A? B! A? B!
A? B! A? B! A?

\loop for start loop, <code> inside, then \if<..>-family, another bunch of <code>, ended with \repeat.

For loop

```
\usepackage{forloop}
\newcounter{themenumber}
\forloop{themenumber}{1}{
↪   \value{themenumber} < 5}{
    A?
}
```

A? A? A? A?

\usepackage{forloop}

```
\usepackage{pgffor}
\foreach \n in {0,...,4}{
    \n\space
}
\foreach \n in {apples,burgers,cake}{
    Let's eat \n.\par
}
```

0 1 2 3 4
Let's eat apples.
Let's eat burgers.
Let's eat cake.

\usepackage{pgffor}, part of pgf, part of TikZ

https://stackoverflow.com/questions/2561791/iteration-in-latex

Reqursion

```
\def\requr#1{\ifnum#1>0 A?
↪  \requr{\numexpr#1 - 1 }\fi}
\requr{8}
```

A? A? A? A? A? A? A? A?

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

“let” command

```
Run one:\\
\def\a{I'm first macro}
\def\b{\a}
\def\a{I'm the second macro}
\a, \b

Run two:\\
\def\a{I'm first macro}
\let\b=\a
\def\a{I'm the second macro}
\a, \b
```

Run one:
I’m the second macro, I’m the second macro
Run two:
I’m the second macro, I’m first macro

The statement “`\let`\a=\b” gives \a the current meaning of \b. If \b changes after the assignment is made, \a does not change.

Usecase with \let: “decorator”

Imagine: you have some \command used inside the document multiple times. You want to *add some addition behaviour* to the command – decorate (or “wrap”, or “redefine with the use of itself”). You can do it with \let:

```
\let\oldCommand=\command
\def\command#1{<some code>\oldCommand}
```

And the same for enviruments using \usepackage{etoolbox} or \g@addto@macro

/47351/can-i-redefine-a-command-to-contain-itself /467435/environment-decorators

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

99 Bottles of Beer

```
\newcounter{beer}

\newcommand{\verses}[1]{
  \setcounter{beer}{#1}
  \par\noindent
  \arabic{beer} bottles of beer on the
  ↪  wall,\\
  \arabic{beer} bottles of beer!\\
  Take one down, pass it around---\\
  \addtocounter{beer}{-1}
  \arabic{beer} bottles of beer on the
  ↪  wall!\\
  \ifnum#1>0
    \verses{\value{beer}}
  \fi
}

\begin{document}
\verses{99}
\end{document}
```

99 bottles of beer on the wall,
99 bottles of beer!
Take one down, pass it around—
98 bottles of beer on the wall!

98 bottles of beer on the wall,
98 bottles of beer!
Take one down, pass it around—
97 bottles of beer on the wall!

97 bottles of beer on the wall,
97 bottles of beer!
Take one down, pass it around—
96 bottles of beer on the wall!

96 bottles of beer on the wall,
96 bottles of beer!
Take one down, pass it around—
95 bottles of beer on the wall!

95 bottles of beer on the wall,
95 bottles of beer!
Take one down, pass it around—

https://rosettacode.org/wiki/99_Bottles_of_Beer#LaTeX

not-AND logical gate

```
\newcommand{\nand}[2]{
\ifnum #1=#2
    \ifnum#1=1
        0
    \else
        1
    \fi
\else
    1
\fi
}

\begin{tabular}{cc|c}
A & B & not-and\\\hline
0 & 0 & \nand{0}{0}\\
1 & 0 & \nand{1}{0}\\
0 & 1 & \nand{0}{1}\\
1 & 1 & \nand{1}{1}\\
\end{tabular}
```

A	B	not-and
0	0	1
1	0	1
0	1	1
1	1	0

Split words

```
\def\testwords#1{%
    \begingroup
    \edef\tempa{#1\space}%
    \expandafter\endgroup
    \expandafter\readwords\tempa\relax
}
\def\readwords#1 #2\relax{%
      \doword{#1}%  #1 = substr, #2 = rest of string
      \begingroup
      \ifx\relax#2\relax  % is #2 empty?
         \def\next{\endgroup\endtestwords}% your own
        ↪   end-macro if required
      \else
         \def\next{\endgroup\readwords#2\relax}%
      \fi
      \next
}
\def\doword#1{(#1)}
\def\endtestwords{}

\testwords{Now good enough}\\
\testwords{Now good}
```

(Now)(good)(enough)
(Now)(good)

What we will know?

- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff

Filesystem writing

```
\newwrite\filereg
\openout\filereg=outfile.txt\relax
    \write\filereg{{\noexpand\bf Hello}\
    ↪  World}                                  {\bf Hello}\ World
     \write\filereg{{\noexpand\bf              {\bf byebye}\ World
     ↪  byebye}\ World}
\closeout\filereg

\lstinputlisting{outfile.txt}
```

The output in pdf is the result of listing the outfile.txt

- `\newwrite` to get a free register
- `\openout` to open filename for output
- `\write`<register> to actually write
- `\noexpand` to stop expanding (also see `\expandafter`)
- `\closeout` to close the file

You also can use numbers. Like `\write`4

Filesystem reading

```
\newread\filereg
\openin\filereg=infile.txt\relax

\ifeof\filereg file ended \else file continue \fi
\read\filereg to\myline
\myline\par
\ifeof\filereg file ended \else file continue \fi
\read\filereg to\myline
\myline\par
\ifeof\filereg file ended \else file continue \fi
\read\filereg to\myline
\myline\par
\ifeof\filereg file ended \else file continue \fi
\closein\filereg
```

file continue **Hello** World
file continue **byebye** World
file continue
file ended

- `\newread` to get a free register
- `\openin` to open filename for input
- `\read`<register> `to`\<newvariable> to actually read
- `\ifeof` to check if file is still have lines
- `\closein` to close the file

You also can use numbers. Like `\read`4

How BiBLaTeX works? Proof-of-concept

```
\newwrite\filetowrite
\openout\filetowrite=\jobname.xxxx\relax
\def\setInfoFromEnd#1{\write\filetowrite{{#1
↪  \the\count0}}}

\newread\filefromread
\openin\filefromread=\jobname.xxxx\relax
\def\readWhileNotEof{
\ifeof\filefromread
    \closein\filefromread
\else
    \read\filefromread to\newline
    \newline~\\
    \readWhileNotEof
\fi
}

\readWhileNotEof{}

\setInfoFromEnd{A\textit{A}A \textbf{bb}}
↪  \vspace*{\fill}\newpage
\setInfoFromEnd{another reference}
```

AAA **bb** 1
another reference 2

1. write info into a file
2. use an external command to do something with the file
3. read content from a file in a different place

Use command line

```
\immediate\write18{wget
↪  https://www.google.ru/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png
↪  -O image.png}
```


```
\includegraphics[scale=0.1]{image.png}
```

Use

- `\write`18 to call the command line
- `\immidiate` to run it as it reached (otherwise only when TEX will "print" the page)
- use `--enable-write18 -interaction=nonstopmode` keys for run offline
- the commands with internet connection will not work at Papeeria

/20444/what-are-immediate-write18-and-how-does-one-use-them. Also look at `filecontents` env.

Command names manipulation

```
\csname textit\endcsname{Italic}
\string\textit
```

Italic \textit

- `\csname` — `\endcsname` to “compile” command from name
- `\string` to show the name

Catcodes

```
{
\catcode`\[=1 \catcode`\]=2
\catcode`\{=12 \catcode`\}=12
\catcode`\#=12
a#b
[{aasd}]
]
```

a#b {aasd}

`\catcode` shows what symbol will be responsible for the group, what for comment etc.

Expanding mechanism

- TEX support overriding practically everything
- You need a detail mechanism sometimes: so you want to run a command, print it or just put it deeper in macros
- `\expandafter` expands the token after the next one before the next token itself.
- `\noexpand` Prevents the next token from being expanded.

/Articles/How_does_%5Cexpandafter_work%3A_The_meaning_of_expansion
/Articles/How_does_%5Cexpandafter_work%3A_A_detailed_macro_case_study

LuaTEX Proof-of-concept

```
\usepackage{luacode}

A random number:
\begin{luacode}
tex.print(math.random())
\end{luacode}
```

A random number: 0.67535939611278

What we will know?

Technical agreements

Simple command creation and style files creation

Programming

Debugging

What we will know?

- Debugging
  - Show-family
  - Tracing-family
  - Other debugging ways

Expand macros

```
\def\testT#1{my tex macro #1}
\newcommand{\testL}[1]{my latex macro #1}
\let\testLe\testL
\def\testMult#1: #2{#2\testT{#1}}

\show\testL
\show\testT
\show\testLe
\show\testMult
```

```
> \testL=\long macro:
#1->my latex macro #1.
l.38 \show\testL

> \testT=macro:
#1->my tex macro #1.
l.39 \show\testT

> \testLe=\long macro:
#1->my latex macro #1.
l.40 \show\testLe

> \testMult=macro:
#1: #2->#2\testT {#1}.
l.41 \show\testMult
```

`\show<macros>`

Expand macros

`show` with not macros

```
\newlength{\mylengthL}
\setlength{\mylengthL}{2ex}
\newdimen\mylengthT
\mylengthT=0.4em
\show\mylengthL
\show\mylengthT
```

```
> \mylengthL=\skip49.
1.43 \show\mylengthL

> \mylengthT=\dimen147.
1.44 \show\mylengthT
```

Show length and counts

```
\newlength{\mylengthL}
\setlength{\mylengthL}{2ex}
\newdimen\mylengthT
\mylengthT=0.4em

\newcounter{mycountL}
\setcounter{mycountL}{67}
\newcount\mycountT
\mycountT=50

\showthe\mylengthL
\showthe\mylengthT

\showthe\mycountT
\makeatletter
\showthe\c@mycountL
\makeatother
\showthe<var>
```

```
> 8.62pt.
l.39 \showthe\mylengthL

> 3.99994pt.
l.40 \showthe\mylengthT

> 50.
l.42 \showthe\mycountT

> 67.
l.44 \showthe\c@mycountL
```

Show boxes

```
\newsavebox{\boxname}
\savebox{\boxname}{LaTeX content}
\newbox\mybox
\setbox\mybox=\hbox{TeX content}

\showboxdepth=100
\showboxbreadth=100
\showbox\mybox
\showbox\boxname
```

```
> \box42=
\hbox(6.83+0.10999)x53.88
.\EU1/lmr/m/n/10 TeX
.\glue 3.33 plus 1.66331 minus 1.1111
.\EU1/lmr/m/n/10 content

! OK.
l.38 \showbox\mybox


> \box41=
\hbox(6.83+0.10999)x65.13
.\EU1/lmr/m/n/10 LaTeX
.\glue 3.33 plus 1.66331 minus 1.1111
```

`\showbox<box>`

https://texfaq.org/FAQ-printvar #showbox-rp #showboxbreadth-rp #showboxdepth-rp kn: 11

`\showboxbreadth` and `\showboxdepth` provides how much info to show

Show-family list

`\show` log macros insides

`\showthe` log length or counter value

`\showbox` log box insides

- `\showboxdepth` the value of the deepest level of box nesting
- `\showboxbreadth` the maximum number of items shown per level

`\showlists` writes the content of partial box lists in all of the 4 non-math TeX modes

`\showhyphens{W}` displays the hyphenation of W on the terminal/log according to the hyphenation rules.

What we will know?

- Debugging
  - Show-family
  - Tracing-family
  - Other debugging ways

Trace modes and commands

```
\def\testC#1{#1\hbox{oo}\vbox{z}}
\tracingcommands=1
\testC{B}
\tracingcommands=0
```

```
{horizontal mode: the letter B}
{\hbox}
{restricted horizontal mode: the
    letter o}
{end-group character }}
{horizontal mode: \vbox}
{internal vertical mode: the letter z
    }
{horizontal mode: the letter z}
{end-group character }}
{blank space  }
{\tracingcommands}
```

\tracingcommands=1

Trace macros (recursively)

```
\def\testT#1{my tex macro #1}
\newcommand{\testL}[1]{my latex macro #1}
\let\testLe\testL
\def\testMult#1: #2{#2\testT{#1}}
\tracingmacros=1
\testL{A}
\testT{B}
\testLe{V}
\testMult{d}: {Ki}
\tracingmacros=0
```

```
\testL #1->my latex macro #1
#1<-A

\testT #1->my tex macro #1
#1<-B

\testLe #1->my latex macro #1
#1<-V

\testMult #1: #2->#2\testT {#1}
#1<-d
#2<-Ki

\testT #1->my tex macro #1
#1<-d
```

\tracingmacros=1

Tracing-family list

`\tracingcommands` if positive, writes commands to the .log file

`\tracinglostchars` if positive, writes characters not in the current font to the .log file

`\tracingmacros` if positive, writes to the .log file when expanding macros and arguments

`\tracingonline` if positive, writes diagnostic output to the terminal as well as to the .log file

`\tracingoutput` if positive, writes contents of shipped out boxes to the .log file

`\tracingpages` if positive, writes the page-cost calculations to the .log file

`\tracingparagraphs` if positive, writes a summary of the line-breaking calculations to the .log file

`\tracingrestores` if positive, writes save-stack details to the .log file

`\tracingstats` if positive, writes memory usage statistics to the .log file

`\tracingall` turns on every possible mode of interaction

What we will know?

- Debugging
  - Show-family
  - Tracing-family
  - Other debugging ways

Message to log file

```
\def\testT#1{my tex macro #1}
\newdimen\mylengthT
\mylengthT=0.4em
\newcounter{mycountL}
\setcounter{mycountL}{67}
\message{Message text: \the\mylengthT\
↪  \themycountL\ \testT{d}}
\typeout{typeout text: \the\mylengthT\
↪  \themycountL\ \testT{d}}
\mylengthT=0.4em

\newcounter{mycountL}
\setcounter{mycountL}{67}
\newcount\mycountT
\mycountT=50
```

```
Message text: 3.99994pt\ 67\ my tex
    macro d
typeout text: 3.99994pt\ 67\ my tex
    macro d
```

`\message{<msg>}` – TEX-command, `\typeout{<msg>}` – LATEX-command

What we have learned today?

- Technical agreements
- Simple command creation and style files creation
  - Command creation
  - Style and class files
- Programming
  - Define macros
  - Conditions
  - Loops and recursion
  - Related things to macros creation
  - Programming examples
  - Relatate stuff
- Debugging
  - Show-family
  - Tracing-family
  - Other debugging ways

references I

color from the footnotes corresponds to references' color.

- kn: Knuth "The TEXBook"
- lv: L'vovsky "Nabor i verstka v sisteme LATEX"
- lamport: Lamport. "LATEX. A Document Preparation System, User's Guide and Reference Manual"
- man: "LATEX2e: An unofficial reference manual" also at website https://latexref.xyz/
- : https://tex.stackexchange.com/questions
- : https://en.wikibooks.org/wiki/LaTeX
- : https://www.overleaf.com/learn/latex
- : https://www.tug.org/utilities/plain/cseq.html

Distribution

- the pdf-version of the presentation and all printed materials can be distributed under license Creative Commons Attribution-ShareAlike 4.0 https://creativecommons.org/licenses/by-sa/4.0/
- The source code of the presentation is available on https://github.com/Lavton/latexLectures and can be distributed under the MIT license https://en.wikipedia.org/wiki/MIT_License#License_terms