LaTeX: from dummy to TeXnician

Command creation

Anton Lioznov

Skoltech

ISP 2019,
lesson 5

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

Agreements

I

Footnotes

- Only in the “out-class” version
- For second reading
- Containe advanced usage of the command
- Containe references to read more
  - to the exact chapter
  - (often) with the href to exact page
- Containe some comments

Like this

Agreements

II

Addition information – “magic”

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

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

Why?

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

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

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}
\vspace*{\fill}\vspace{-5ex}
\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!
$\frac{\partial f}{\partial x}$

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

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

man: 12.1 lv: VII.1.1 /Macros /Commands#Defining_a_new_command

Recreate new command

```latex
\newcommand{\lookAtMe}{\vbox{I'm
    mister Meeseeks look at me!}}
\newcommand{\dfdx}{\ensuremath{\frac
    {\partial f}{\partial x}}}

\renewcommand{\lookAtMe}{\vbox{I'm
    missis Meeseeks look at me!}}
\begin{document}
\lookAtMe \lookAtMe \lookAtMe \
    lookAtMe
\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!
$\frac{\partial f}{\partial x}$

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

`\renewcommand` to recreate already created command

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}
\vspace*{\fill}\vspace{-5ex}
\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!

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

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

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

to define command with star use \@ifstar man: 12.4

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`

man: 12.8

Where to put command creation

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? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

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:

```
\DeclareOption{<option>}{<code>}
```

like:

```
\DeclareOption{a4paper}{%
\setlength{\paperheight}{297mm}%
\setlength{\paperwidth}{210mm}%
}
```

Use `\ProcessOptions` after all option declaration!

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}
```

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? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

Entities

1. Primitive commands
2. Macros
3. Counters (=integer numbers)
4. Lengths
5. Glues
6. Boxes
7. Strings

We already see something about all this stuff except counters. Let us look at them! (And then return to look deeper at the boxes, lengths, glues and strings)

What we will know?

Counters etc

- Counters
- Length
- Skips
- Toks
- Boxes

What is “counter”

“Counter” is just an integer number.
It’s using in multiple places to count everything in LaTeX: sections, equations, references, citation, enumerate lists,...

Define and simple manipulation with counters

```latex
\newcounter{abcd}
\arabic{abcd}

\setcounter{abcd}{2017}
\arabic{abcd}

\addtocounter{abcd}{-27}
\arabic{abcd}
```

0
2017
1990

- \newcounter to define new counter
- \setcounter to set counter to new value
- \addtocounter to add a number to the counter

lv: VII.3.1 man: 12.5 man: 13.4-5 LaTeX/Counters#Counter_manipulation

Print counter

`\arabic{countname}`	1	2	3	4	5	6	7	8	9
`\alph{countname}`	a	b	c	d	e	f	g	h	i
`\Alph{countname}`	A	B	C	D	E	F	G	H	I
`\roman{countname}`	i	ii	iii	iv	v	vi	vii	viii	ix
`\Roman{countname}`	I	II	III	IV	V	VI	VII	VIII	IX
`\fnsymbol{countname}`	*	†	‡	§	¶	‖	**	††	‡‡

P.S. `\value` to get "raw" value of the counter

//en.wikibooks.org/wiki/LaTeX/Counters#Counter_style *13.1[126]*
see lv: IX.2.3 *for russian analog of* `\alph`

pre-defined counters in standart classes

part chapter section subsection subsubsection
paragraph subparagraph

page figure table footnote equation
enumi enumii enumiii enumiv

TEX's counters (will talk later) `\year` `\month`
`\day` `\time`

Counters#LaTeX_default_counters man: 13

Counter Domination

problem

You may want to write something like

1 Action

Task #1.1. Prepare the template
Task #1.2. Write the code
Task #1.3. Look at it

2 Viewing

Task #2.1. Compile the code 1.2

But the straightforward solution will give you

1 Action

Task #1. Prepare the template
Task #2. Write the code
Task #3. Look at it

2 Viewing

Task #4. Compile the code 1

Counter Domination

straightforward solution

```
\newcounter{task}
\newcommand{\tsk}{\par\addtocounter{
    task}{1}%
\textbf{Task \#\arabic{task}.} }

\section{Action}
\tsk Prepare the template
\tsk Write the code \label{write}
\tsk Look at it
\section{Viewing}
\tsk Compile the code \ref{write}
```

1 Action

Task #1. Prepare the template
Task #2. Write the code
Task #3. Look at it

2 Viewing

Task #4. Compile the code 1

Counter Domination

The Way

\newcounter{task} → \newcounter{task}[section]

\newcounter{<slave>}[<master>] will resets the value of <slave> if the value of <master> is change

\addtocounter{task}{1} → \refstepcounter{task}

\refstepcounter{<counter>} use it to update \label–\ref mechanism

\textbf{Task \#\arabic{task}. → \textbf{Task \#\arabic{section}.\arabic{task}.

Inside \newcommand{\tsk} to redefine the labels

→ \renewcommand{\thetask}{\arabic{section}.\arabic{task}}

\renewcommand{\the<counter>} to redefine the reference

lv: VII.3.3 man: 13.6

Counter Domination

solution

```
\newcounter{task}[section]
\newcommand{\tsk}{\par\refstepcounter
    {task}%
\textbf{Task \#\arabic{section}.\
    arabic{task}.} }
\renewcommand{\thetask}{\arabic{
    section}.\arabic{task}}
\section{Action}
\tsk Prepare the template
\tsk Write the code \label{write}
\tsk Look at it
\section{Viewing}
\tsk Compile the code \ref{write}
```

1 Action

Task #1.1. Prepare the template
Task #1.2. Write the code
Task #1.3. Look at it

2 Viewing

Task #2.1. Compile the code 1.2

Redefine existing counter domination

"equation" example

Package based solution:

```
\usepackage{chngcntr}
```

and `\counterwith{equation}{chapter}` to make the "equation" a slave or `\counterwithout{equation}{chapter}` to "free" the counter.

Core-based solution:

```
\makeatletter
\@removefromreset{equation}{section}
\@addtoreset{equation}{chapter}
\renewcommand{\theequation}{\thechapter.\@arabic\c@equation}
\makeatother
```

/61756/how-to-change-equation-numbering-style

/54241/change-the-type-of-equation-numbering-in-document-class-article

/28333/continuous-v-per-chapter-section-numbering-of-figures-tables-and-other-docume

https://texfaq.org/FAQ-running-nos lv: IX.2.1.

Also see `\p@` prefix lv: IX.2.2 and /61426/how-to-make-ref-display-only-subsection

Define and simple manipulation

Define new `\newcount\<countname>` as `\newcount\mycounter`
Set number `\<countname>=<number>` Or use `\countdef`. Like `\countdef\mynumber=43`
Add number `\advance\<countname> by <number>`. Also there are `\multiply` and `\divide`. As well as `\numexp`.
Show number `\the\<countname>` or `\number` or `\romannumeral`

kn: 15 /245635/formal-syntax-rules-of-dimexpr-numexpr-glueexpr
Actually you can use `\count<number>` like `\count212`. What `\newcount` do is just find a free number and fix it to your defined name.

Define and simple manipulation

Example

```
\newcount\counttest
\counttest=40
\advance\counttest by 2
\the\counttest

\number\year\ is \romannumeral\year\
    par
```

42
2019 is mmxix

What we will know?

Counters etc

- Counters
- Length
- Skips
- Toks
- Boxes

Length manipulation

Define lenght `\newdimen\<lenname>`
Set length `\stringlenname>=<len>`
Add lenght `\advance\<lenname> by <len>`. Also there are `\multiply` and `\divide`. As well as `\dimexp`.
Show lenght `\the\<lenname>`

Length manipulation

Example

```
\newdimen\mylen
\mylen=40mm
\advance\mylen by 2cm
\the\mylen
```

170.71652pt

Length manipulation

Define lenght `\newlength{\<lenname>}`
Set length `\setlength`
Add lenght `\addtolength`.
Show lenght `\the\<lenname>`. But also you can use `\usepackage{printlen}` and then `\uselengthunit`, `\printlength`

Length manipulation

Example

```latex
\usepackage{printlen}
\newlength{\mylen}
\setlength{\mylen}{40mm}
\addtolength{\mylen}{2cm}
\the\mylen

\uselengthunit{mm}\printlength{\mylen
    }
```

170.71652pt
59.99928 mm

What we will know?

Counters etc

- Counters
- Length
- Skips
- Toks
- Boxes

Skip manipulation

TEX provides additional storage for glue and glue in math mode (that is sensible for math style). They have the same syntax as length, just with `\skip` or `\muskip` prefix/suffix

What we will know?

Counters etc

- Counters
- Length
- Skips
- Toks
- Boxes

Toks manipulation

TEX has addition registers for storing strings. They have `\toks` prefix/suffix. The difference between toks and storage inside macros are in extension...

What we will know?

Counters etc

- Counters
- Length
- Skips
- Toks
- Boxes

Box manipulation

Define box `\newbox\<boxname>`
Set box `\setbox\<boxname>=<box>`
Provide the content without deleting it: `\copy\<boxname>`
Provide the content with delete from memory:
`\box\<boxname>`
Dimentions: width: `\wd`, height: `\ht`, depth: `\dp`

Box manipulation

Example

```
\newbox\mybox
\setbox\mybox=\hbox{TeX content}

the box width \the\wd\mybox
the box height \the\ht\mybox
the box depth \the\dp\mybox

provide the content: \copy\mybox
provide the content and free the box:
     \box\mybox
```

the box width 53.88pt
the box height 6.83pt
the box depth 0.10999pt
provide the content: TeX content
provide the content and free the box: TeX content

`\the` here is just to PRINT the length. Not the part of the command.

Box manipulation

Define box `\newsavebox{\<boxname>}`
Set box `\savebox`
Provide the content without deleting it: `\usebox`
Dimentions:

1. Create a length variable: `\newlength`
2. Set the variable to dimention of the box CONTENT:
   - width: `\settowidth{\<len-var>}{\usebox{\<box>}}`
   - height: `\settoheight{\<len-var>}{\usebox{\<box>}}`
   - depth: `\settodepth{\<len-var>}{\usebox{\<box>}}`

man: 14.4-6 man: 20.5, 20.7

Box manipulation

Example

```
\newsavebox{\mybox}
\savebox{\mybox}{\hbox{LaTeX content
    }}

\newlength{\boxwidth}
\settowidth{\boxwidth}{\usebox{\mybox
    }}
\newlength{\boxheight}
\settoheight{\boxheight}{\usebox{\
    mybox}}
\newlength{\boxdepth}
\settodepth{\boxdepth}{\usebox{\mybox
    }}

the box width \the\boxwidth
the box height \the\boxheight
the box depth \the\boxdepth

provide the content: \usebox{\mybox}
```

the box width 65.13pt
the box height 6.83pt
the box depth 0.10999pt
provide the content: LaTeX content

`\the` here is just to PRINT the length. Not the part of the command.

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

What we will know?

Programming

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

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

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

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

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>]`

Only "=", ">" or "<" are allowed.

Use `\ifcase` to check different stuff.

Also use `\ifodd` to check if num is odd or even

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

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 XeLaTeXby \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

What we will know?

Programming

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

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.

Reqursion

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

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

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

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

What we will know?

Programming

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

“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.

kn: 20 *https://www.tug.org/utilities/plain/cseq.html#let-rp*

Usecase with \let: “decorator”

Imagine: you have some \commandused 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

99 Bottles of Beer

```latex
\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

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

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)

/12810/how-do-i-split-a-string/12811

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

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}
```

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

Use \usepackage{pgfkeys}

New package with key=value syntax

```
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{packexample}[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}
```

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

hello?

```
\usepackage[myarg]{packexample}
\showarg
```

defaultarg

```
\usepackage{packexample}
\showarg
```

noarg

What we will know? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

Debugging

Filesystem writing

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

\lstinputlisting{outfile.txt}
```

```
{\bf Hello}\ World
{\bf byebye}\ World
```

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 `\write4`

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}
```

A*A*A **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
    }
```

Google

Use

- `\write18` 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.

https://www.tug.org/utilities/plain/cseq.htmlcatcode-rp kn: app. B

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? I

Introduction

Simple command creation

Simple .sty and .cls file creation

Counters etc

Programming

What we will know? II

LaTeX's advanced creation

Relatate stuff

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.
l.43 \show\mylengthL

> \mylengthT=\dimen147.
l.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
```

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

> 3.99994pt.
l.40 \showthe\mylengthT

> 50.
l.42 \showthe\mycountT

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

\showthe<var>

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

```latex
\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}}
```

```
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

#message-rp kn: 20 man: 27.2

What we have learned today? I

What we have learned today? II

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

LaTeX's advanced creation

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

references II

- : https://www.overleaf.com/learn/latex
- : https://www.tug.org/utilities/plain/cseq.html
- http://ctan.altspu.ru/info/macros2e/macros2e.pdf list of internal LaTeX macros
- https://www.latex-project.org/help/documentation/clsguide.pdf guide for class and package writers

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