Code-Presentations

LaTeX Ninja

June 2019

Introduction

Intro i

When I thought about it, I realized there are actually so many ways of displaying code using LaTeX packages. So I’ll start with the most basic and then go on to the more advanced ones ;)

Intro ii

texttt

```
\texttt{}
```

This one isn’t a verbatim way to express code, but it will change the font to typewriter, so it ’looks like code’. However, in these short bits of code, you will have to use escape sequences for reserved characters.
I make massive use of when ’talking’ about code or, you know, writing explanatory text sequences. It is especially useful for bits of code where there is no excessive (or none at all) use of escape sequences. Then it is a really handy way to quickly typeset code. Once you have lots of reserved characters, you might be better of just using this next one.

Intro iii

The verbatim environment

```
begin{verbatim}
code goes here
end{verbatim}
```

This one really is a staple and pretty failsafe, but also doesn’t have code highlighting which you might want in most cases, apart from very short bits of code where highlighting isn’t important. You can use verbatim as an environment for multiple lines of code which appear like a quote as a separate block in your text. In other cases, where you just want but without having to escape reserved characters, you might want to use the `some code` command. You can use any characters as delimiters to denote beginng and end of code. So it can also be `test`. The idea is that you can choose one which you will not need inside the code, as not to ’confuse’ the enviroment.

Bugs

1. listing all the bugs
2. oops, this already is one of the main bugs:
    - If you were to use the `enumitem` package
    - you would get a fatal error
    - but no output
    - due to package conflicts

Tcblisting

Tcblisting

The code is quite tiny because I set the font size to very small in the definition of the little `myr`

Load files

```r
# text <- readLines(file.choose())
filePath <- "http://www.link.com/a_text.txt"
text <- readLines(filePath)
```

lstlisting

Using Code Listings

```
my_vector <- c("testing","vectors")
my_vector # a test
```

Using Code Listings II

This time with caption but without code highlighting.

Listing 1: Hello World! in C

```
int main()
{
    printf("Hello World!");
    return 0;
}
```

Code highlighting with listings

Listing 2: A demonstration

```
import numpy as np
```

You can also use the inline shorthand for small snippets: `while{$a || $b}`

List of listings

Well, this is how it’s supposed to be, but sadly, using sections inside frames will add up to this result. – So you can’t use this list of listings here.

Importing listing from file

```
<codelisting ref="#test">
   <fun>
      <bla>
         lalalalala <lb/>
         <!-- comment -->
      </bla>
   </fun>
</codelisting>
```

This will only be in colour if you use the settings.

Using Knitr

```
# Create sequence
my_sequence = 1:5

# use summary function to display stats
summary(my_sequence)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##       1       2       3       3       4       5
```

Then output values inside the text: 1, 2, 3, 4, 5.

Using Knitr with options

```r
# This package (gutenbergr) is not available
# on Overleaf
# So we set KnitR with eval=FALSE
# so it will not evaluate
# and litter our slides with error messages
library(gutenbergr)

# echo=FALSE hides the code but displays the results
# echo=1:3 only displays first three lines
# background=#FFFFFF
```

Using `minted`

```c
int main() {
  printf("hello, world");
  return 0;
}
```

`Can also typeset $code$ inline. Yay to \LaTeX{}!`

Using `minted`

There also is the option to include math mode stuff in the comments.

```
/*
π = lim_{n→∞} P_n/d
*/
const double pi = 3.1415926535;
```

Using `minted` with options

```python
import numpy as np

test = 5
```

This is it ☺