1 The Pygment\TeX{} package

This document demonstrates how to use the Pygment\TeX{} package to typeset code listings with L\LaTeX{} and Pygments\footnote{http://pygments.org/}.

Pygments is a generic syntax highlighter for general use in all kinds of software such as forum systems, wikis or other applications that need to prettify source code.

Pygment\TeX{} provides an environment and two commands for typesetting code listings in a \LaTeX{} document:

- the \texttt{pygmented} environment typesets its contents as a source code listing,
- the \texttt{\inputpygmented} command typesets the contents of a file, including the result in the \LaTeX{} document, and
- the \texttt{\pyginline} command typesets its contents, keeping the result in the same line.

They accept many options that allow the user to configure the listing in many ways.

Read the remaining of this document to have an idea of what the package is capable of.

2 Requirements

Current versions of Pygment\TeX{} require Python version 3. Earlier versions required Python version 2. All versions require the Python Pygments library.

3 How to use the package

1. Add the package to the document preamble.

\texttt{\usepackage{pygmentex}}

2. Use the environment or commands mentioned previously to include source code listings on your document.
3. Compile using pdflatex\textsuperscript{2} All the source code listings in the document will be collected and saved in a temporary file with the extension .snippets in its name.

4. Run pygmentex documentname.snippets. The python application pygmentex is distributed with the Pygmentex package. It will produce another temporary file with the extension .pygmented, containing \LaTeX{} code for the code listings previously collected. The next time the document is compiled, they are included to produce the final typeset document.

5. Rerun pdflatex as usual.

Note. Running the external python application pygmentex can be done automatically from within \LaTeX{} if the optional argument \texttt{--shell-escape} is used. With

\texttt{pdflatex --shell-escape <file>}

the external run of pygmentex is not needed. It will be run automatically at the end of the document compilation.

There is a package option \texttt{force} that will force running pygmentex every time the document is compiled.

4 First examples

A simple verbatim text is the first example.

\begin{verbatim}
Hello world!
This is a simple demonstration text.
\end{verbatim}

The following C program reads two integers and calculates their sum.

\begin{verbatim}
#include <stdio.h>
int main(void)
{
    int a, b, c;
    printf("Enter two numbers to add: ");
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("Sum of entered numbers = %d\n", c);
    return 0;
}
\end{verbatim}

\textsuperscript{2}Other \LaTeX{} compilers may also work but have not been tested by the author.
# include <stdio.h>
int main(void)
{
    int a, b, c;
    printf("Enter two numbers to add: ");
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("Sum of entered numbers = %d\n", c);
    return 0;
}

In this program, `int` is a type and "Enter two numbers to add: " is a literal string.

Next you can see a Java program to calculate the factorial of a number.

```java
public class Factorial
{
    public static void main(String[] args)
    {
        int number = 5;
        int factorial = 1;
        for (int i = 1; i <= number; i++)
            factorial = factorial * i;
        System.out.println("Factorial of " + number + " is " + factorial);
    }
}
```

5 Options

5.1 lang

The programming language of the listing code can be specified using the `lang` option.

To get a list of all available languages, execute the following command on the command line:

```
$ pygmentize -L lexers
```

5.2 sty

Instead of using the default style you may choose another stylesheet provided by Pygments by its name using the `sty` option.
To get a list of all available stylesheets, execute the following command on the command line:

```
$ pygmentize -L styles
```

Creating your own styles is also very easy. Just follow the instructions provided on the website.

As examples you can see a C program typeset with different styles.

```c
#include<stdio.h>
main()
{
  int n;
  printf("Enter a number: ");
  scanf("%d",&n);
  if ( n%2 == 0 )
    printf("Even\n");
  else
    printf("Odd\n");
  return 0;
}
```

5.3 font

The value of the option font is typeset before the content of the listing. Usually it is used to specify a font to be used. See the following example.
object bigint extends Application {
  def FACTORIAL(N: BigInt): BigInt = 
    if (N == 0) 1 else N * FACTORIAL(N-1)

  val F50 = FACTORIAL(50); val F49 = FACTORIAL(49)
  println("50! = " + F50)
  println("49! = " + F49)
  println("50!/49! = " + (F50 / F49))
}

5.4 colback

The option colback can be used to choose a background color, as is shown in the following example.

let rec factorial n = 
  if n = 0 then 1 
  else n * factorial (n - 1)
System.Console.WriteLine(factorial anInt)

5.5 gobble

The option gobble specifies the number of characters to suppress at the beginning of each line (up to a maximum of 9). This is mainly useful when environments are indented (Default: empty no character suppressed).
6 if(n < 2)  
7 return 1;  
8 else  
9 return n * fact(n - 1);  
10}  
11\end{pygmented}
12\end{minipage}

A code snippet inside a minipage:

\begin{pygmented}[lang=common-lisp,linenos,linenostart=1001,linenostep=2,linenosep=5mm]
;; Triple the value of a number
(defun triple (X)
  "Compute three times X."
  (* 3 X))
\end{pygmented}

5.6 tabsize

The option \texttt{tabsize} specifies the number of spaces given by a tab character (Default: 8).

5.7 linenos, linenostart, linenostep, linenosep

The lines of a listing can be numbered. The followig options control numbering of lines.

- Line numbering is enabled or disable with the \texttt{linenos} boolean option.
- The number used for the first line can be set with the option \texttt{linenostart}.
- The step between numbered lines can be set with the option \texttt{linenostep}.
- The space between the line number and the line of the listing can be set with the option \texttt{linenosep}.

In the followig listing you can see a Scheme function to calculate the factorial of a number.
(define (list-of-squares n)
  (let loop ((i n) (res '()))
    (if (< i 0)
      res
      (loop (- i 1) (cons (* i i) res))))))

5.8 caption and label

The option caption can be used to set a caption for the listing. The option label allows the assignment of a label to the listing.

Here is an example:

```
\begin{pygmented}[lang=c++,label=lst:test,caption=A \textbf{C++} example]
// This program adds two numbers and prints their sum.
#include <iostream>
int main()
{
  int a;
  int b;
  int sum;
  sum = a + b;
  std::cout << "The sum of " << a << " and " << b << " is " << sum << "\n";
  return 0;
}
\end{pygmented}
```

Listagem 1: A C++ example

```cpp
// This program adds two numbers and prints their sum.
#include <iostream>
int main()
{
  int a;
  int b;
  int sum;
  sum = a + b;
  std::cout << "The sum of " << a << " and " << b << " is " << sum << "\n";
  return 0;
}```
Listing \ref{lst:test} is a C++ program.

Listing 1 is a C++ program.

5.9 texcomments, mathescape and escapeinside

The option texcomments, if set to \texttt{true}, enables LaTeX comment lines. That is, LaTeX markup in comment tokens is not escaped so that LaTeX can render it.

The \texttt{mathescape}, if set to \texttt{true}, enables LaTeX math mode escape in comments. That is, $...$ inside a comment will trigger math mode.

The option escapeinside, if set to a string of length two, enables escaping to LaTeX. Text delimited by these two characters is read as LaTeX code and typeset accordingly. It has no effect in string literals. It has no effect in comments if texcomments or mathescape is set.

Some examples follows.

\begin{pygmented}[lang=c++,texcomments]
#include <iostream>

using namespace std;

main()
{
    cout << "Hello World"; // prints \underline{Hello World}
    return 0;
}
\end{pygmented}

\begin{pygmented}[lang=python,mathescape]
# Returns $\sum_{i=1}^{n}i$

def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
\end{pygmented}

\begin{pygmented}[lang=c,escapeinside=||]
if (condition)
{\texttt{command$_1$}}
else
{\texttt{command$_2$}}
\end{pygmented}
\begin{verbatim}
if (condition)
    command1
else
    command2
\end{verbatim}

5.10 inline method and boxing method

After being prettified by Pygments, the listings are enclosed in a command (for \texttt{pyginline}) or in an environment (for \texttt{pygmented} and \texttt{inputpygmented}). By default, \texttt{pyginline} uses the command \texttt{\efbox} from the \texttt{efbox} package, and \texttt{pygmented} and \texttt{inputpygmented} use the environment \texttt{mdframed} from the \texttt{mdframed} package.

The enclosing command or environment should be configurable using a list of key-value pairs written between square brackets.

The enclosing command for \texttt{pyginline} can be changed with the option \texttt{inline method}. For instance, in the following the command \texttt{\tcbox} from the \texttt{tcolorbox} package is used:

\begin{verbatim}
\begin{pyginline}[lang=java,inline method=tcbbox]
"Factorial of "] is a literal string.
\end{pyginline}
\end{verbatim}

In the previous Java program, \texttt{"Factorial of "} is a literal string.

The enclosing environment for \texttt{pygmented} and \texttt{inputpygmented} can be changed with the option \texttt{boxing method}. For instance, here is a hello world program in C#, enclosed in a \texttt{tcolorbox} environment:

\begin{verbatim}
\begin{pygmented}[lang=csharp,boxing method=tcolorbox]
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
\end{pygmented}
\end{verbatim}

\begin{verbatim}
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
\end{verbatim}

Any option unknown to PygmentsTex are passed to the enclosing command or environment.

For instance:

\begin{verbatim}
\begin{inputpygmented}
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
\end{inputpygmented}
\end{verbatim}
6 Setting global options for Pygment\TeX

Global options can be setting using the \texttt{setpygmented} command. See the examples that follows.

\begin{pygmented}{lang=haskell, colback=red!30, font=\ttfamily\small}
\begin{verbatim}
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
\end{verbatim}
\end{pygmented}

\begin{pygmented}{lang=snobol}
\begin{verbatim}
OUTPUT = "What is your name?"
\end{verbatim}
\end{pygmented}
Username = INPUT
OUTPUT = "Thank you, " Username
END

\begin{pygmented}
OUTPUT = "What is your name?"
Username = INPUT
OUTPUT = "Thank you, " Username
END
\end{pygmented}
See the identifier \texttt{variable}, which names something. String literals in C looks like \\
\texttt{"hello, world!\n"}.

This one \texttt{let x = [1;2;3] in length x} is an OCaml expression with local bindings. With OCaml one can do imperative, functional and object oriented programming.

Now some Java code: \texttt{public int f(double x)}. This is a method header.

8 More examples of displayed code snippets

In listing 2 you can see a function definition in the Scheme language. This function computes the factorial of a natural number.

\begin{lstlisting}[language=Scheme,escapechar=!]
(define fact
 (lambda (n)
 (if (= n 0)
 1
 (* n (fact (- n 1)))))
\end{lstlisting}

Here you have some more code to further testing the package. Listing 3 is a Haskell program. When run this program interacts with the user asking the user name, reading a line input by the user, and showing a greeting message to the user.

\begin{lstlisting}[language=Haskell,escapechar=!]
Listagem 3: A haskell interactive program
\end{lstlisting}
This is a rule:

Now a Pascal procedure:

```pascal
procedure example(a: integer);
const
  A = 'jeja';
var
  sMessage: string;
begin
  ShowMessage(sMessage + A);
end.
```

and a Pascal program

```pascal
Program HelloWorld(output)
var
  msg : String
begin
  msg = 'Hello, world!';
  Writeln(msg)
end.
```

A Python code snippet:

```python
# -*- coding: utf-8 -*-
def parse_opts(dic, opts):
    for opt in re.split(r'\s*\s*\,', opts):
        x = re.split(r'\s*\s*\,', opt)
        if len(x) == 2 and x[0] and x[1]:
            dic[x[0]] = x[1]
        elif len(x) == 1 and x[0]:
            dic[x[0]] = True
    return dic
```
9 Using code snippets in environments

The following is a description environment.


```scala
def qsort(xs: List[Int]): List[Int] = 
  xs match {
    case Nil => Nil
    case pivot :: tail =>
      qsort(tail filter { _ < pivot }) :::
      pivot :: qsort(tail filter { _ >= pivot })
  }
```


```python
function entry0 (o)
    N=N + 1
    local title = o.title or '(no title)'
    fwrite('<LI><A HREF="#%d">%s</A>

end
```


10  A long program

Here you can read the source code for a hand written lexical analyser for the straight-line programming language that I have developed in Java.

<table>
<thead>
<tr>
<th>Ad hoc lexical analyser</th>
</tr>
</thead>
<tbody>
<tr>
<td>import java.io.IOException;</td>
</tr>
<tr>
<td>import java.io.Reader;</td>
</tr>
<tr>
<td>import java.util.Hashtable;</td>
</tr>
<tr>
<td>import java.util.Map;</td>
</tr>
<tr>
<td>public class Lexer</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>private Reader in;</td>
</tr>
<tr>
<td>private int x;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>private Map&lt;String,Token.T&gt; reserved =</td>
</tr>
<tr>
<td>new Hashtable&lt;String,Token.T&gt;();</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>public Lexer(Reader in) throws IOException</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.in = in;</td>
</tr>
<tr>
<td>x = in.read();</td>
</tr>
<tr>
<td>reserved.put(&quot;let&quot;, Token.T.LET);</td>
</tr>
<tr>
<td>// acrescentar demais palavras reservadas</td>
</tr>
<tr>
<td>// ...</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>public Token get() throws IOException</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>// retornar o próximo símbolo léxico do programa</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>while (Character.isWhitespace(x))</td>
</tr>
<tr>
<td>x = in.read();</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (x == -i)</td>
</tr>
<tr>
<td>return new Token(Token.T.EOF);</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if ((char)x == ','</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>x = in.read();</td>
</tr>
<tr>
<td>return new Token(Token.T.COMMA);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>if (Character.isDigit(x))</td>
</tr>
<tr>
<td>{</td>
</tr>
</tbody>
</table>
StringBuilder builder = new StringBuilder();
builder.append((char)x);
while (Character.isDigit((x = in.read())))
    builder.append((char)x);
return new Token(Token.T.INT, new Long(builder.toString()));
}

if (Character.isAlphabetic(x))
{
    StringBuilder builder = new StringBuilder();
    builder.append((char)x);
    while (Character.isAlphabetic(x = in.read()) ||
            Character.isDigit(x) || (char)x == '_')
        builder.append((char)x);
    String s = builder.toString();
    Token.T t = reserved.get(s);
    if (t == null)
        return new Token(Token.T.ID, s);
    return new Token(t);
}

// completar demais tokens

System.out.println("unexpectec char: <" + (char)x + ">");
x = in.read();
return get();
}

### 11 Some fancy examples using tcolorbox

The followig example uses tcolorbox to typeset the code listing.

**Example 1: hello from Scala**

```scala
object HelloWorld extends App {
    println("Hello, world!")
}
```

**My fancy title**

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!")
    }
}
```
module Main (main) where

main :: IO ()
main = putStrLn "Hello, world!"

#include <iostream>
using namespace std;

int main(int argc, char** argv) {
    cout << "Hello, world!" << endl;
    return 0;
}

/* This program prints a
hello world message
to the console. */

import std.stdio;

void main()
{
    writeln("Hello, World!");
}

12 Some fancy examples using mdframed
The followig example uses mdframed to typeset the code listing.

with Ada.Text_IO;

procedure Hello_World is
use Ada.Text_IO;
begin
    Put_Line("Hello, world!");
end;

Saying hello from Pascal

program HelloWorld;
WriteLn('Hello, world!');

Saying **hello** in Modula-2

```modula-2
MODULE Hello;
FROM STextIO IMPORT WriteString;
BEGIN
  WriteString("Hello World!");
END Hello.
```

Exercise n1

```go
// hello world in 'go'
package main

import "fmt"

func main() {
  fmt.Println("Hello, world!")
}
```

Exercise n2

```objc
/* hello from objective-c */
#import <stdio.h>
#import <Foundation/Foundation.h>

int main(void) {
  NSLog(@"Hello, world!\n");
  return 0;
}
```

Hello from C

```c
#include <stdio.h>
int main(int argc, char **argv) {
  printf("Hello, world!\n");
  return 0;
}
```
13 Debugging

Paths given to `\inputpygmented` should be relative to the top level project directory, not to the file that contains the command (if that file is in a subdirectory). `PygmenteX` generates only a single top-level `.snippets` file, and paths are not munged to account for code in subdirectories.

14 Conclusion

That is all.