Testing the Pygmen\TeX{} package
José Romildo Malaquias
August 12, 2014

1 The Pygmen\TeX{} package

This document demonstrates how to use the Pygment\TeX{} package to typeset code listings with \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.

Pygmen\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{includepygmented} 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 How to use the package

In order to use the package, start by putting

\begin{verbatim}
\usepackage{pygmentex}
\end{verbatim}

in the preamble of the document.

Use the environment or commands mentioned previously to include source code listings on your document.

When compiling the document (with pdflatex, for instance), all the source code listings in the document will be collected and saved in a temporary file with the extension \texttt{.snippets} in its name. Then the auxiliary program pygmentex (a Python application distributed with the Pygmen\TeX{} package) should be run taking this file as input. It will produce another temporary file with the extension \texttt{.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.

The programming language of the listing code can be specified using the \texttt{lang} option.

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

\texttt{$ pygmentize -L lexers }

\section{First examples}

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

\begin{pygmented}[lang=c]
\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}
\end{pygmented}

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

In this program, \texttt{int} is a type and \texttt{"Enter two numbers to add: "} is a literal string.

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

\begin{inputpygmented}[lang=java]{Factorial.java}
public class Factorial
{
  public static void main(String[] args)
  {
    int number = 5;
    int factorial = 1;
  
  \end{inputpygmented}

In this program, \texttt{int} is a type and \texttt{"Enter two numbers to add: "} is a literal string.

Next you can see a Java program to calculate the factorial of a number.
for (int i = 1; i <= number; i++)
    factorial = factorial * i;
System.out.println("Factorial of " + number + " is " + factorial);
}

4 Choosing different Pygments styles

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.
# 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 Choosing a 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.

1 \begin{pygmented}[lang=scala,font=\rmfamily\scshape\large]
2 object bigint extends Application {
3  def factorial(n: BigInt): BigInt =
4     if (n == 0) 1 else n * factorial(n-1)
5     val f50 = factorial(50); val f49 = factorial(49)
6     println("50! = " + f50)
7     println("49! = " + f49)
8     println("50!/49! = " + (f50 / f49))
9 }
10 \end{pygmented}

6 Changing the background color

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

1 \begin{pygmented}[lang=fsharp,colback=green!25]
2 let rec factorial n =
3    if n = 0 then 1
4    else n * factorial (n - 1)
11 \end{pygmented}
let rec factorial n =
  if n = 0
  then 1
  else n * factorial (n - 1)
System.Console.WriteLine(factorial anInt)

7 Supressing initial characters
The option \texttt{gobble} specifies the number of characters to suppress at the be-
ginning of each line (up to a maximum of 9). This is mainly useful when
environments are indented (Default: empty — no character suppressed).

A code snippet inside a minipage:
\begin{minipage}[t]{.5\linewidth}
\begin{pygmented}[lang=d,gobble=8]
ulong fact(ulong n)
{
  if(n < 2)
  return 1;
  else
  return n * fact(n - 1);
}
\end{pygmented}
\end{minipage}

A code snippet inside a minipage:
\begin{pygmented}[lang=common-lisp,tabsize=4]
;; Triple the value of a number
(defun triple (X)
  "Compute three times \texttt{X}.
  \((3 \times X)\)"
  (* 3 X))
\end{pygmented}

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

\begin{pygmented}[lang=common-lisp,tabsize=4]
;; Triple the value of a number
(defun triple (X)
  "Compute three times \texttt{X}.
  \((3 \times X)\)"
  (* 3 X))
\end{pygmented}
Compute three times X.
(* 3 X)

9 Numbering lines

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

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

In the following 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))))
```

10 Captioning

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:

```
// This program adds two numbers and prints their sum.
#include <iostream>
int main()
{
  int a;
```
# 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;
}

Listagem 1: A C++ example

Listing \ref{lst:test} is a C++ program.

11 Escaping to \LaTeX inside a code snippet

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

The `mathescape`, if set to `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}
```cpp
#include <iostream>
using namespace std;

main()
{
    cout << "Hello World"; // prints Hello World
    return 0;
}
```

```python
# Returns $\sum_{i=1}^{n}i$

def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
```

```c
if (condition)
    command$_1$
else
    command$_2$
```

### 12 Enclosing command and environment

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

```python
# In the previous Java program,
\texttt{pyginline}[lang=java,inline method=tcbbox]="Factorial of ", is a
\texttt{literal string.}
```
In the previous Java program, "Factorial of " is a literal string.

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

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

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

For instance:

```
<!-- This is a note -->
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don’t forget me this weekend!</body>
</note>
```

13 Setting global options for PygmenteX

Global options can be setting using the `setpygmented` command. See the examples that follows.
```
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs

elem :: Eq a => a -> [a] -> Bool
elem _ [] = False
elem x (y:ys) = x == y || elem x ys
```

```
OUTPUT = "What is your name?"
Username = INPUT
OUTPUT = "Thank you, " Username
```

```
Module Module1
    Sub Main()
        Console.WriteLine("Hello, world!")
    End Sub
End Module
```
14 More examples of inline code snippets

An inline source code snippet:
```c
const double alfa = 3.14159;
```
This is a C declaration with initialization.

An inline source code snippet:
```prolog
avo(A,B) :- pai(A,X), pai(X,B).
```
is a Prolog clause. Its head is `avo(A,B)` and its body is `pai(A,X), pai(X,B)`.

See the identifier `variable`, which names something. String literals in C looks like "hello, world!"

This one
```ocaml
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:
```java
public int f(double x)
```
is a method header.

```java
public int f(double x)
```
is a method header.
15 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.

**Listagem 2: A Scheme function.**

```scheme
(define fact
  (lambda (n)
    (if (= n 0)
      1
      (* n (fact (- n 1))))))
```

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.

**Listagem 3: A haskell interactive program.**

```haskell
module Main where

-- the main IO action
main = do { putStr "What is your name? ", name''' <- read
          , putStrLn ("Hello, " ++ name''')
    }
```

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
Program HelloWorld(output)

```pascal
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*=', 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
```

16 Using code snippets in environments

The following is a description environment.


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


13
aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.


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


17 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.

```
Ad hoc lexical analyser

import java.io.IOException;
import java.io.Reader;
import java.util.Hashtable;
import java.util.Map;

public class Lexer
{
    private Reader in;
    private int x;

    private Map<String,Token.T> reserved =
        new Hashtable<String,Token.T>();

    public Lexer(Reader in) throws IOException
    {
```
this.in = in;
x = in.read();
reserved.put("let", Token.T.LET);
// acrescentar demais palavras reservadas
// ... }

public Token get() throws IOException
{
    // retornar o próximo símbolo léxico do programa
    while (Character.isWhitespace(x))
        x = in.read();

    if (x == -1)
        return new Token(Token.T.EOF);

    if (((char)x == ',')
    {
        x = in.read();
        return new Token(Token.T.COMMA);
    }

    if (Character.isDigit(x))
    {
        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("unexpected char: "+ (char)x + ");
    x = in.read();
    return get();
}
Some fancy examples using \texttt{tcolorbox}

The following example uses \texttt{tcolorbox} to typeset the code listing.

\textbf{Example 1: hello from Scala}

\begin{tcolorbox}
\texttt{object HelloWorld extends App { 
  println("Hello, world!")
}}
\end{tcolorbox}

\begin{tcolorbox}
\texttt{public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, world!")
  }
}}
\end{tcolorbox}

\begin{tcolorbox}
\texttt{module Main (main) where}

\begin{verbatim}
main :: IO ()
main = putStrLn "Hello, world!"
\end{verbatim}
\end{tcolorbox}

\begin{tcolorbox}
\texttt{# include \textless iostream\textgreater 
using namespace std;
int main(int argc, char** argv) {
  cout << "Hello, world!" << endl;
  return 0;
}}
\end{tcolorbox}

\begin{tcolorbox}
\texttt{/* This program prints a 
  hello world message 
  to the console. */}

\begin{verbatim}
import std.stdio;

void main()
{
  writeln("Hello, World!");
}
\end{verbatim}
\end{tcolorbox}
19 Some fancy examples using \texttt{mdframed}

The following example uses \texttt{mdframed} to typeset the code listing.

```ada
with Ada.Text_IO;

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

Saying \textit{hello} from Pascal

```pascal
program HelloWorld;
begin
   WriteLn('Hello, world!');
end.
```

Saying \textit{hello} in Modula-2

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

// hello world in 'go'
```go
package main
import "fmt"
func main() {
   fmt.Println("Hello, world!")
}
```

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

Exercise n1

Exercise n2

10points
20 Conclusion

That is all.