Testing the PygmenTEX package

José Romildo Malaquias
malaquias@ufop.edu.br
`https://github.com/romildo/pygmentex`

March 18, 2026

1 The PygmenTEX package

This document demonstrates how to use the PygmentTEX package to typeset code listings with LATEX and Pygments[1].

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.

PygmenTEX provides an environment and two commands for typesetting code listings in a LATEX document:

- the `pygmented` environment typesets its contents as a source code listing,
- the `\inputpygmented` command typesets the contents of a file, including the result in the LATEX document, and
- the `\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 PygmenTEXrequire 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.

```
\usepackage{pygmentex}
```

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

[1] `http://pygments.org/`

3. Compile using `pdflatex`.[2] All the source code listings in the document wil 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 withing LATEX if the optional argument `-shell-escape` is used.

With

```
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 `force` that will force running `pygmentex` every time the document is compiled.

4 First examples

A simple verbatim text is the first example.

```
\begin{pygmented}[]
Hello world!
  This is a simple demonstration text.
\end{pygmented}
```

```
Hello world!
  This is a simple demonstration text.
```

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

```
\begin{pygmented}[lang=c]
#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{pygmented}
```

[2]Other LATEXcompilers may also work but have not been tested by the author.

```c
#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, \pyginline[lang=c]|int| is a type and
\pyginline[lang=c]|"Enter two numbers to add: "| is a literal string.
```

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.

```
\inputpygmented[lang=java]{Factorial.java}
```

```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.

```
\noindent
\begin{minipage}[t]{0.49\linewidth}
  \begin{pygmented}[lang=c,gobble=4,sty=murphy]
    #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;
    }
  \end{pygmented}
\end{minipage}
\hfil
\begin{minipage}[t]{0.49\linewidth}
  \begin{pygmented}[lang=c,gobble=4,sty=trac]
    #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;
    }
  \end{pygmented}
\end{minipage}
```

```
#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;
}
```

```
#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. Usualy it is used to specify a font to be used. See the following example.

```
\begin{pygmented}[lang=scala,font=\rmfamily\scshape\large]
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))
}
\end{pygmented}
```

```
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 folowing example.

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

```
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).

```
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:

```
ulong fact(ulong n)
{
  if(n < 2)
    return 1;
  else
    return n * fact(n - 1);
}
```

5.6 autogobble

The boolean option `autogobble` automatically removes common leading whitespace from the code snippet. This is highly recommended when your `pygmented` environments are indented to match the surrounding LaTeX source code, as it saves you from having to manually count and specify the exact number of characters to suppress using the standard `gobble` option.

```
A code snippet deeply indented in the \LaTeX{} source:
\begin{itemize}
  \item Item one
  \item Item two
        \begin{pygmented}[lang=python,autogobble,colback=yellow!20]
          def greet(name):
              # The indentation relative to the 'def' keyword is preserved!
              print(f"Hello, {name}!")
              return True
        \end{pygmented}
\end{itemize}
```

A code snippet deeply indented in the LaTeX source:

- Item one
- Item two

```
def greet(name):
    # The indentation relative to the 'def' keyword is preserved!
    print(f"Hello, {name}!")
    return True
```

5.7 tabsize

The option `tabsize` specifies the number of of spaces given by a tab character (Default: 8).

```
\begin{pygmented}[lang=common-lisp,tabsize=4]
;; Triple the value of a number
```

```
(defun triple⇥(X)
⇥"Compute three times X."
⇥(* 3 X))
\end{pygmented}
```

```
;; Triple the value of a number
(defun triple   (X)
    "Compute three times X."
    (* 3 X))
```

5.8 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 `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 followig listing you can see a Scheme function to calculate the factorial of a number.

```
\begin{pygmented}[lang=scheme,linenos,linenostart=1001,linenostep=2,linenosep=5mm]
;; Building a list of squares from 0 to 9.
;; Note: loop is simply an arbitrary symbol used as
;; a label. Any symbol will do.

(define (list-of-squares n)
  (let loop ((i n) (res '()))
    (if (< i 0)
        res
        (loop (- i 1) (cons (* i i) res)))))
\end{pygmented}
```

```
1001  ;; Building a list of squares from 0 to 9.
      ;; Note: loop is simply an arbitrary symbol used as
1003  ;; a label. Any symbol will do.

1005  (define (list-of-squares n)
        (let loop ((i n) (res '()))
1007      (if (< i 0)
              res
1009          (loop (- i 1) (cons (* i i) res)))))
```

5.9 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*

```
// 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.

5.10 texcomments, mathescape and escapeinside

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}
```

```
#include <iostream>
using namespace std;
main()
{
   cout << "Hello World";  // prints Hello World
   return 0;
}
```

```
\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}
```

```
# Returns $\sum_{i=1}^{n} i$
def sum_from_one_to(n):
    r = range(1, n + 1)
    return sum(r)
```

```
\begin{pygmented}[lang=c,escapeinside=||]

if (|\textit{condition}|)
    |\textit{command$_1$}|
else
    |\textit{command$_2$}|
\end{pygmented}
```

```
if (condition)
    command_1
else
    command_2
```

5.11 inline method and boxing method

After being prettified by Pygments, the listings are enclosed in a command (for `\pyginline`) or in an environment (for `pygmented` and `inputpygmented`). By default `\pyginline` uses the command `\efbox` from the `efbox` package, and `pygmented` and `inputpygmented` use the environment `mdframed` from the `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 `\pyginline` can be changed with the option `inline method`. For instance, in the following the command `\tcbox` from the `tcolorbox` package is used:

```
 In the previous Java program,
 \pyginline[lang=java,inline method=tcbox]|"Factorial of "| is a
 literal string.
```

In the previous Java program, `"Factorial of "` is a literal string.

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

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

```
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:

```
\begin{pygmented}[lang=xml,boxing method=tcolorbox,colframe=red,boxrule=2mm]
<!-- This is a note -->
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
\end{pygmented}
```

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

6 Setting global options for PygmenTEX

Global options can be setting using the `setpygmented` command. See the examples that follows.

```
\setpygmented{lang=haskell, colback=red!30, font=\ttfamily\small}

\begin{pygmented}[]
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
\end{pygmented}
```

```haskell
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
```

```
\begin{pygmented}[colback=blue!20, boxing method=tcolorbox]
elem :: Eq a => a -> [a] -> Bool
elem _ [] = False
elem x (y:ys) = x == y || elem x ys
\end{pygmented}
```

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

```
\setpygmented{lang=snobol}

\begin{pygmented}[]
          OUTPUT = "What is your name?"
          Username = INPUT
          OUTPUT = "Thank you, " Username
END
\end{pygmented}
```

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

```
\setpygmented{test/.style={colback=yellow!33,boxing method=tcolorbox,colframe=blue}}

\begin{pygmented}[test, lang=vbnet]
Module Module1
    Sub Main()
        Console.WriteLine("Hello, world!")
    End Sub
End Module
\end{pygmented}
```

```vbnet
Module Module1
    Sub Main()
        Console.WriteLine("Hello, world!")
    End Sub
End Module
```

```
\begin{pygmented}[lang=tcl]
puts "Hello, world!"
\end{pygmented}
```

```
puts "Hello, world!"
```

7 More examples of inline code snippets

```
An inline source code snippet:
\pyginline[lang=c]|const double alfa = 3.14159;|.
This is a C declaration with initialization.
```

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

```
\pyginline[lang=prolog,colback=yellow]=avo(A,B) :- pai(A,X), pai(X,B).=
is a Prolog clause. Its head is
\pyginline[lang=prolog,sty=emacs,colback=yellow,linecolor=red]=avo(A,B)=
and its body is
\pyginline[lang=prolog,sty=vim,colback=black,hidealllines]=pai(A,X), pai(X,B)=.
```

`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 \pyginline[inline method=efbox,colback=green!25]|variable|,
which names something. String literals in C looks like
\pyginline[lang=c,inline method=tcbox,colback=blue!20,boxrule=2pt]|"hello, world!\n"|.
```

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

```
This one
\pyginline[lang=ocaml,font=\ttfamily\scriptsize,topline=false]: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.
```

This one `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:
\pyginline[lang=java,sty=colorful,font=\ttfamily\itshape,linewidth=1pt]|public int f(double x)|.
This is a method header.
```

Now some Java code: *`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.

Listagem 2: *A Scheme function.*

```
(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*

```
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:

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

and a Pascal program

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

A Python code snippet:

```
# -*- 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.

An item Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

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

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class 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.

Another item Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

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

Nullam eleifend justo in nisl. In hac habitasse platea dictumst. Morbi nonummy. Aliquam ut felis. In velit leo, dictum vitae, posuere id, vulputate nec, ante. Maecenas vitae pede nec dui dignissim suscipit. Morbi magna. Vestibulum id purus eget velit laoreet laoreet. Praesent sed leo vel nibh convallis blandit. Ut rutrum. Donec nibh. Donec interdum. Fusce sed pede sit amet elit rhoncus ultrices. Nullam at enim vitae pede vehicula iaculis.

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.

***Ad hoc* lexical analyser**

```java
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("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`

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

My fancy title

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

```
module Main (main) where

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

My title

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
  cout << "Hello, world!" << endl;
  return 0;
}
```

My title

```
/* 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;

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

Saying *hello* in Modula-2

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

Exercise n1

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

import "fmt"

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

Exercise n2 **10points**

```
/* 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). PygmenTEXgenerates only a single top-level `.snippets` file, and paths are not munged to account for code in subdirectories.

14 Conclusion

That is all.