pandoc-minted

A pandoc filter to render LaTeX code blocks using minted

Usage

```
pandoc [OPTIONS] --filter pandoc-minted [FILES]
```

Source

As usual, declare a module `Main`, and `import` some useful definitions, `intercalate` from `Data.List`, `topDown` from `Text.Pandoc.Generic`, and everything from `Text.Pandoc.JSON`.

```
module Main where

import           Data.List           (intercalate)
import           Text.Pandoc.Generic (topDown)
import           Text.Pandoc.JSON    (Attr, Block(..), Format(..), Inline(..),
                                      Pandoc, toJSONFilter)
```

The `Minted` Data Type

Define a data type `Minted` to more expressively handle inline code and code blocks.

```
data Minted
  = MintInline (String, String) String
  | MintedBlock (String, String) String
```

Define a `Show` instance for `Minted` in order to generate LaTeX code.

```
instance Show Minted where
  show (MintInline (attrs, language) contents)  =
      "\\mintinline[" ++ attrs ++ "]{" ++ language ++ "}{" ++ contents ++ "}"
  show (MintedBlock (attrs, language) contents) =
      unlines [ "\\begin" ++ "{minted}[" ++ attrs ++ "]{" ++ language ++ "}"
              , contents
              , "\\end" ++ "{minted}"
              ]
```

The `main` Function

Run `minted` as a JSON filter.

```
main :: IO ()
main = toJSONFilter minted

minted :: Maybe Format -> (Pandoc -> Pandoc)
minted (Just (Format "latex")) = topDown (concatMap mintinline) .
                                 topDown (concatMap mintedBlock)
minted _                       = id
```

Handle Inline Code

Transform *Code* into a `\mintinline` call, otherwise return a given *Inline*.

```
mintinline :: Inline -> [Inline]
mintinline (Code attr contents) =
    let
      latex = MintInline (unpackCode attr "text") contents
    in
    [ RawInline (Format "latex") (show latex) ]
mintinline x                    = [x]
```

Handle Code Blocks

Transform a *CodeBlock* into a `minted` environment, otherwise return a given *Block*.

```
mintedBlock :: Block -> [Block]
mintedBlock (CodeBlock attr contents) =
    let
        latex = MintedBlock (unpackCode attr "text") contents
    in
        [ RawBlock (Format "latex") (show latex) ]
mintedBlock x                         = [x]
```

Helper Functions

Given a triplet of `Attr`ibutes (identifier, language(s), and key/value pairs) and a default language, return a pair of `minted` attributes and language.

```
unpackCode :: Attr -> String -> (String, String)
unpackCode (_, [], kvs) defaultLanguage = (unpackAttrs kvs, defaultLanguage)
unpackCode (identifier, "sourceCode" : "literate" : language : _, kvs) _ =
    (unpackAttrs kvs, language)
unpackCode (identifier, "sourceCode" : language : _, kvs) _ =
    (unpackAttrs kvs, language)
unpackCode (_, language : _, kvs) _     = (unpackAttrs kvs, language)
```

Given a list of key/value pairs, return a string suitable for `minted` options.

```
unpackAttrs :: [(String, String)] -> String
unpackAttrs kvs = intercalate ", " [ k ++ "=" ++ v  | (k, v) <- kvs ]
```