```
module Pdf
  -- document part
  ( createPdf  -- :: PageSize -> [Page] -> String
  , createPage -- :: [Area] -> Page
  , A4         -- :: PageSize
  , pageSize   -- :: PageSize -> (Int, Int, Int, Int)

  -- area part
  , position -- :: Block -> Float -> Float -> Area

  -- block part
  , textBlock  -- :: [Command] -> Block
  , over       -- :: Block -> Block -> Block
  , under      -- :: Block -> Block -> Block
  , rightAlign -- :: Block -> Block

  -- commands part
  , Helvetica -- :: Font
  , Courier   -- :: Font
  , Times     -- :: Font
  , Arial     -- :: Font
  , setFont   -- :: String -> Int -> Command
  , write     -- :: String -> Command
  , writeLn   -- :: String -> Command
  ) where
-----------------------------------------------------------------------
-- the datatypes
type Object = String
data Command
  = Td Int Int
  | Tf String Int
  | TL Int
  | Quote String
  | Tj String
  | Tc Int
  | Tw Int
  | Tr Int
  | Tz Int
  | G Int
  | T'
  | Cm Float Float Float Float Float Float
  | BT
  | ET
  | Qq
  | Q
data Block = Block [Command]
type Area = (Float, Float, Block)
type Page = [Area]
data Font
  = Helvetica
  | Courier
  | Times
  | Arial
data PageSize = A4

-----------------------------------------------------------------------
-- exported functions
-- document --
createPdf :: PageSize -> [Page] -> String
createPdf size ps = objString ++ (trailer objects xrefOffset)
 where
  (objString, objects, xrefOffset) = arrangeObjects (static ++ pages)
  static = [ objDictionary 1 ["/Type/Catalog", "/Pages 2 0 R"]
           , objDictionary 2 ["/Type/Pages", "/Count " ++ (show (length ps))
                             , "/Kids [" ++ kids ++ "]"]
        , objDictionary 3 ["/ProcSet [/PDF/Text]", "/Font <</F0 4 0 R /F1 5 0
R>>"]
           , objDictionary 4 ["/Type/Font", "/Subtype/Type1"
                             , "/Encoding/WinAnsiEncoding"
                             , "/BaseFont/Helvetica", "/Name/F0" ]
           , objDictionary 5 ["/Type/Font", "/Subtype/Type1"
                             , "/Encoding/WinAnsiEncoding"
                             , "/BaseFont/Courier", "/Name/F1" ]
           , objDictionary 6 ["/Type/Font", "/Subtype/Type1"
                             , "/Encoding/WinAnsiEncoding"
                             , "/BaseFont/Arial", "/Name/F1" ]
           , objDictionary 7 ["/Type/Font", "/Subtype/Type1"
                             , "/Encoding/WinAnsiEncoding"
                             , "/BaseFont/Times", "/Name/F1" ]
           ]
  pages  = translatePages (pageSize size) 8 ps
  kids   = unwords (take (length ps) (map (\n -> (show (n)) ++ " 0 R") [8,11..
]))

  trailer :: Int -> Int -> String
  trailer n off = "trailer\n<<\n\t/Size " ++ (show n) ++
    "\n\t/Root 1 0 R\n>>\nstartxref\n" ++
 (show off) ++ "\n%%EOF"

createPage :: [Area] -> Page
createPage = id
pageSize :: PageSize -> (Int, Int, Int, Int)
pageSize A4 = (0, 0, 595, 842)
pageSize _  = error "Unknown pagesize"
-- area --
position :: Block -> Float -> Float -> Area
position b x y = (x, y, b)
-- block --
textBlock :: [Command] -> Block
textBlock = Block
over :: Block -> Block -> Block
over (Block b) (Block b') = Block (b++(Quote "":b'))
under :: Block -> Block -> Block
under b b' = over b' b
rightAlign :: Block -> Block
rightAlign (Block [])     = Block []
rightAlign (Block (b:bs)) = case b of
  Tj s      -> Block ((shifted s) ++ rest)
  Quote s   -> Block ((shifted s) ++ (T':rest))
  otherwise -> Block (b:rest)
 where
  Block rest = rightAlign (Block bs)
  shifted s  =
 ( (G 1)
    : (Tz (-100))
 : (Tj s)
    : (Tz 100)
    : (G 0)
    : [Tj s]
 )
-- commands --
setFont :: Font -> Int -> Command
setFont Helvetica size = Tf "/F0" size
setFont Courier size   = Tf "/F1" size
```

```haskell
setFont Arial size     = Tf "/F2" size
setFont Times size     = Tf "/F3" size
write :: String -> Command
write s = Tj s
writeLn :: String -> Command
writeLn s = Quote s
---------------------------------------------------------------------
-- local helper functions
-- creates an object
obj :: Int -> [String] -> Object
obj n ss = (show n) ++ " 0 obj\n" ++ (unlines (map ('\t':) ss)) ++ "endobj"
-- creates an object containing a dictionary
objDictionary :: Int -> [String] -> Object
objDictionary n ss =
  (show n) ++ " 0 obj\n<<\n" ++
  (unlines (map ('\t':) ss)) ++ ">>\nendobj"
-- creates an object containing a stream
objStream :: Int -> [String] -> (Object, Int)
objStream n ss =
  ((show n) ++ " 0 obj\n\t<< /Length " ++ (show (n+1)) ++
   " 0 R >>\nstream\n" ++ ss' ++ "endstream\nendobj"
  , length ss')
 where
  ss' = unlines (map ('\t':) ss)
-- the pdf-header
header :: String
header = "%PDF-1.3\n%äãÏÒ"
-- concats all objects and add a xref at the end
arrangeObjects :: [Object] -> (String, Int, Int)
arrangeObjects os = (content ++ xref, objLen, 2 + length content)
 where
  content   = header ++ concat objects
  xref      = "\nxref\n0 " ++ (show objLen) ++ "\n" ++ refs
  objLen    = 1 + length objects
  objects   = map ('\n':) os
  refs      = unlines ("0000000000 65535 f ":(map makeRef positions))
  makeRef   = (\p -> (reverse (offset p)) ++ " 00000 n ")
  positions = calcPos ((length header) + 2) (map ((1 +) . length) os)
  offset    = (\p -> take 10 ((reverse (show p))++(concat (repeat "0"))))

  calcPos :: Int -> [Int] -> [Int]
  calcPos _ []         = []
  calcPos start (x:xs) = (start):(calcPos (start+x) xs)
{-
-- findFonts :: [Page] -> [String]
findFonts ps = map (\(Tf f _) -> f) fonts
 where
  fonts = filter (\x -> case x of Tf _ _ -> True; _ -> False) commands
  commands = concat (concat (map (map (\(_,_,c) -> c)) ps))
-}
-- translates all pages to objects
translatePages :: (Int, Int, Int, Int) -> Int -> [Page] -> [Object]
translatePages s@(x,y,w,h) n ps = concat (map makePageObjs pageCommands)
 where
  pageCommands = zip [n, (n+3)..] (map (translatePage s) ps)
  makePageObjs :: (Int, [Command]) -> [Object]
  makePageObjs (n,cs) =
    [ objDictionary n ["/Type/Page", "/Parent 2 0 R", "/Resources 3 0 R"
         , "/Mediabox [" ++ (show x) ++ " " ++ (show y) ++
   " " ++ (show w) ++ " " ++ (show h) ++ "]", "/Contents [" ++
   (show (n+1)) ++ " 0 R]" ]
 , content
 , obj (n+2) [show (len)]
 ]
   where
    (content, len) = objStream (n+1) (translate cs)

-- combines all areas in a page
translatePage :: (Int, Int, Int, Int) -> Page -> [Command]
translatePage s ps =  Q : (concat (map (translateArea s) ps))
-- creates a list of commands from an area
translateArea :: (Int, Int, Int, Int) -> Area -> [Command]
translateArea (sx,sy,sw,sh) =
  (\(x,y,Block b) ->
    (
   Qq :
   (Cm 1 0 0 1 (x-(fromInt sx)) ((fromInt sh)-(y-(fromInt sy)))) :
   (BT:b)
 ) ++ [ET])
-- translates commands to text
translate :: [Command] -> [String]
translate []             = []
translate (Td x y:cs)  = ((show x) ++ " " ++ (show y) ++ " Td") : (translate c
s)
translate (Tf f s:cs)  = (f ++ " " ++ (show s) ++ " Tf") : ((show s) ++ " TL")
 : (translate cs)
translate (TL l:cs)    = ((show l) ++ " TL") : (translate cs)
translate (Quote s:cs) = ("(" ++ (escape s) ++ ")Tj") : "T*" : (translate cs)
translate (Tj s:cs)    = ("(" ++ (escape s) ++ ")Tj") : (translate cs)
translate (Tc c:cs)    = ((show c) ++ " Tc") : (translate cs)
translate (Tw w:cs)    = ((show w) ++ " Tw") : (translate cs)
translate (Tr r:cs)    = ((show r) ++ " Tr") : (translate cs)
translate (Tz z:cs)    = ((show z) ++ " Tz") : (translate cs)
translate (G g:cs)     = ((show g) ++ " g") : (translate cs)
translate (T':cs)      = "T*" : (translate cs)
translate (Q:cs)      = "q" : (translate cs)
translate (Qq:cs)      = "Q" : "q" : (translate cs)
translate (BT:cs)      = "BT" : (translate cs)
translate (ET:cs)      = "ET" : (translate cs)
translate (Cm a b c d e f:cs) =
  ((show a) ++ " " ++
   (show b) ++ " " ++
   (show c) ++ " " ++
   (show d) ++ " " ++
   (show e) ++ " " ++
   (show f) ++ " cm") : (translate cs)
-- escape ( ) and \ characters with an extra \
escape :: String -> String
escape []        = []
escape ('(':ss)  = '\\':'(':(escape ss)
escape (')':ss)  = '\\':')':(escape ss)
escape ('\\':ss) = '\\':'\\': (escape ss)
escape (s:ss)    = s:(escape ss)
```