inline — code inlined in a LaTeX document[*]

Jérôme LAURENS[†]

Released 2022/02/07

Abstract

Usually, documentation is put inside the code, inline allows to work the other way round by putting code inside the documentation. This is particularly interesting when different code files share some logic and should be documented all at once.
The file `inline-manual` gives different examples. Here is the implementation of the package.
This LaTeX package requires LuaTeX and may use syntax coloring based on pygment.

1 Package dependencies

luacode, verbatim, datetime2, xcolor, fancyvrb and dependencies of these packages.

2 Similar technologies

The docstrip utility offers similar features, it is somehow more powerful than inline at the cost of more technicality and less practicality,

The ydoc.cls and skdoc.cls are full document classes with similar features but many more that are unrelated. inline focuses on code inlining and interfaces well with pygment for a smart syntax hilighting.

3 Known bugs and limitations

- inline does not play well with docstrip.

4 Presentation

inline is a triptych of three components

1. `inline.sty`
2. `inline-helper.lua`
3. `inline-helper.py`

[*]This file describes version 1.0a, last revised 2022/02/07.
[†]E-mail: jerome.laurens@u-bourgogne.fr

inline.sty mainly declares \NLNCode command and NLN/Code environment. The former allows to insert code chunks as running text whereas the latter allows to instert code snippets as blocks. Moreover, the blocks can be exported to files.

The normal code flow is

1. with inline.sty, LaTeX parses a code snippet, store it in \l_NLN_snippet_tl, and calls either NLN:process_run or NLN:process_block,
2. inline-helper.lua reads the content of some command, and store it in a json file, together with informations to process this code snippet properly,
3. inline-helper.py is asked by inline-helper.lua to read the json file and uses pygment to translate the code snippet into dedicated LaTeX commands. These are stored in a .pyg.tex file named after the md5 digest of the original code chunck, a .pyg.tex LaTeX style file is recorded as well. On return, inline-helper.py gives to inline-helper.lua some LaTeX macros to both input the .pyg.sty and the .pyg.tex file, these are finally executed and the code is displayed with colors.

File I inline-helper.lua implementation

1 Usage

This lua library is loaded by inline.sty with the instruction NLN=require(inline-helper). In the sequel, the syntax to call class methods and instance methods are presented with either a NLN. or a NLN: prefix. Of course either a self. or a self: prefix would be possible, this is what is used in the library for cenveniency.

2 Declarations

```
⟨*lua⟩
local rep  = string.rep
local lpeg = require("lpeg")
local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
local lfs  = require("lfs")
local tex  = require("tex")
require("lualibs.lua")
local json = _ENV.utilities.json
local jobname = token.get_macro('jobname')
```

3 General purpose material

NLN_PY_PATH Location of the inline-helper.py utility.

```
local NLN_PY_PATH = io.popen([[kpsewhich inline-helper.py]]):read('a'):match("^%s*(.-
)%s*$")
```

(*End definition for NLN_PY_PATH. This variable is documented on page* ??.)

`escape` ⟨*variable*⟩ = NLN.escape(⟨*string*⟩)

Escape the given string. NEVER USED?

```
local function escape(s)
    s = s:gsub('\\','\\\\')
    s = s:gsub('\r','\\r')
    s = s:gsub('\n','\\n')
    s = s:gsub('"','\\"')
    return s
end
```

`make_directory` ⟨*variable*⟩ = NLN.make_directory(⟨*string path*⟩)

Make a directory at the given path.

```
local function make_directory(path)
  local mode,_,__ = lfs.attributes(path,"mode")
  if mode == "directory" then
    return true
  elseif mode ~= nil then
    return nil,path.." exist and is not a directory",1
  end
  if os["type"] == "windows" then
    path = path:gsub("/", "\\")
    _,_,__ = os.execute(
      "if not exist "  .. path .. "\\nul " .. "mkdir " .. path
    )
  else
    _,_,__ = os.execute("mkdir -p " .. path)
  end
  mode = lfs.attributes(path,"mode")
  if mode == "directory" then
    return true
  end
  return nil,path.." exist and is not a directory",1
end
local dir_p, json_p = './'..jobname..'.pygd/'
if make_directory(dir_p) == nil then
  dir_p = './'
  json_p = dir_p..jobname..'.pyg.json'
else
  json_p = dir_p..'input.pyg.json'
end
```

`load_exec` NLN.load_exec(⟨*code chunk*⟩)

Class method. Loads the given ⟨*code chunk*⟩ and execute it. On error, messages are printed.

```
local function load_exec(chunk)
  local func, err = load(chunk)
  if func then
    local ok, err = pcall(func)
    if not ok then
      print("inline-helper.lua Execution error:", err)
      print('chunk:', chunk)
    end
  else
    print("inline-helper.lua Compilation error:", err)
    print('chunk:', chunk)
  end
end
```

safe_equals ⟨*variable*⟩ = `NLN.safe_equals(`⟨*string*⟩`)`

Class method. Returns an ⟨=...=⟩ string exactly composed of sufficently many `=` signs such that ⟨*string*⟩ contains neither sequence `[`⟨`=...=`⟩`[` nor `]`⟨*ans*⟩`]`.

```
local eq_pattern = P({ Cp() * P('=')^1 * Cp() + 1 * V(1) })
local function safe_equals(s)
  local i, j = 0
  local max = 0
  while true
    j, i = eq_pattern:match(s, i)
    if j == nil then
      return rep('=', max + 1)
    end
    j = i - j
    if j > max then
      max = j
    end
  end
end
```

options_reset `NLN:options_reset()`
option_add `NLN:option_add(`⟨*string key*⟩`,`⟨*json value*⟩`)`

Instance method. The extra options used for formatting are collected, then forwarded to `inline-helper.py` utility through its JSON input, with key `options`. First we have to clear the option list with `options_reset` before any call to `option_add`.

```
local function options_reset(self)
  self.options = {}
end
local function option_add(self,k,v)
  self.options[k] = v
end
```

start_recording `NLN:start_recording()`

Instance method. In progress.

```
local function start_recording(self)
  self.records = {}
  function self.records.append (t,v)
    t[#t+1]=v
    return t
  end
end
```

load_exec_output NLN:load_exec_output(⟨*code chunk*⟩)

Instance method to parse the ⟨*code chunk*⟩ sring for commands and execute them. The patterns being searched are enclosed within opening `<<<<<` and closing `>>>>>`, each containing 5 characters,

?TEX:⟨*TeX instructions*⟩ the ⟨*TeX instructions*⟩ are executed asynchronously once the control comes back to TEX.

!LUA:⟨*!Lua instructions*⟩ the ⟨*!Lua instructions*⟩ are executed synchronously. When not properly designed, these instruction may cause a forever loop on execution, for example, they must not use `NLN:process_run`.

?LUA:⟨*?Lua instructions*⟩ these ⟨*?Lua instructions*⟩ are executed asynchronously once the control comes back to TEX through a call to `\directlua`, which means that they will wait until any previous asynchronous ⟨*?TeX instructions*⟩ or ⟨*?Lua instructions*⟩ completes.

```
local parse_pattern
do
  local tag = P('?TEX') + '!LUA' + '?LUA'
  local end = '>>>>>'
  local cmd = P(1)^0 - end
  parse_pattern = P({
    '<<<<<' * Cg(tag - ':') * ':' * Cg(cmd) * end * Cp() + 1 * V(1)
  })
end
local function load_exec_output(self, s)
  local i, tag, cmd = 0
  while true do
    tag, cmd, i = parse_pattern:match(s, i)
    if tag == '?TEX' then
      tex.print(cmd)
    elseif tag == '!LUA' then
      self.load_exec(cmd)
    elseif tag == '?LUA' then
      local eqs = self.safe_equals(cmd)
      tex.print([[%
\directlua{self.load_exec([=]]..eqs..[[[]..cmd..[[]=]]..eqs..[[])}%
]])
    else
      return
    end
  end
end
```

process_run NLN:process_run(⟨*cs name*⟩)

Instance method. This is called by function `\NLNCode`. First, we get the content of the ⟨*cs name*⟩ as code to be colored. Then we build a JSON string, save it in a file at `json_p` location. Next we call the `inline-helper.py`, parse its output and execute commands with `load_exec_output`.

```
local function process_run(self, name)
```

```
  if lfs.attributes(json_p,"mode") ~= nil then
    os.remove(json_p)
  end
  local t = {
    ['code']    = token.get_macro(name),
    ['jobname'] = self.jobname,
    ['options'] = self.options or {},
    ['already'] = self.already and 'true' or 'false'
  }
  local s = json.tostring(t,true)
  local fh = assert(io.open(json_p,'w'))
  fh:write(s, '\n')
  fh:close()
  local cmd = "python3 "..NLN_PY_PATH..' "'..\lua_escape:n {json_p}..'"'
  fh = assert(io.popen(cmd))
  self.already = true
  s = fh:read('a')
  self:load_exec_output(s)
end
```

4 Caching

We save some computation time by pygmentizing files only when necessary. The `inline-helper.py` is expected to create a `.pyg.sty` file for a style and a `.pyg.tex` file for colored code. These files are cached during one whole LaTeX run and possibly between different LaTeX runs. Lua keeps track of both the style files created and colored code files created. These tables are populated by a commands in the output of `inline-helper.py` executed synchronously.

`cache_clean_all`
`cache_record`
`cache_clean_unused`

`NLN:cache_clean_all()`
`NLN:cache_record(⟨style name.pyg.sty⟩, ⟨digest.pyg.tex⟩)`
`NLN:cache_clean_unused()`

Instance methods. `cache_clean_all` removes any file in the cache directory `inline.pygd`. This is executed at the beginning of the document processing when there is no aux file. This can be executed on demand with `\directlua{NLN:cache_clean_all()}`. `cache_record` stores both ⟨*style name.pyg.sty*⟩ and ⟨*digest.pyg.tex*⟩. These are file names relative to the ⟨*jobname*⟩`.pygd` directory. `cache_clean_unused` removes any file in the cache directory ⟨*jobname*⟩`.pygd` except the ones that were previously recorded. This is executed at the end of the document processing.

```
local function cache_clean(self)
  local to_remove = {}
  for f in lfs.dir(dir_p) do
    to_remove[f] = true
  end
  for k,_ in pairs(to_remove) do
    os.remove(d .. k)
  end
end
local function cache_record(self, style, colored)
  self.style_set[style] = true
  self.colored_set[colored] = true
```

```
end
local function cache_clean_unused(self)
  local to_remove = {}
  for f in lfs.dir(dir_p) do
    if self.style_set[f] or self.colored_set[f] then
      continue
    end
    to_remove[f] = true
  end
  for k,_ in pairs(to_remove) do
    os.remove(d .. k)
  end
end
```

```
local _DESCRIPTION = [[Global inline helper on the lua side]]
```

5 Return the module

Known fields are

`jobname` to store ⟨*jobname*⟩,

`date` to store ⟨*date string*⟩,

`_VERSION` to store ⟨*version string*⟩,

`dir_p` is the path to the directory where all

Known methods are

escape

make_directory

load_exec

options_reset

option_add

start_recording

process_run

cache_clean_all

cache_record

cache_clean_unused

`pygment` related material is stored,

`json_p` is the path to the JSON file used by `inline-helper.py` utility.

`style_set` the set of style names used

`colored_set` the set of "colored" names used `already` false at the beginning, true after the first call of `inline-helper.py`

```
return {
  _DESCRIPTION       = _DESCRIPTION,
  _VERSION           = token.get_macro('NLNFileVersion'),
  jobname            = jobname,
  date               = token.get_macro('NLNFileDate'),
  NLN_PY_PATH        = NLN_PY_PATH,
  escape             = escape,
  make_directory     = make_directory,
  load_exec          = load_exec,
  options_reset      = options_reset,
  option_add         = option_add,
  start_recording    = start_recording,
  process_run        = process_run,
  cache_clean_all    = cache_clean_all,
  cache_record       = cache_record,
  cache_clean_unused = cache_clean_unused,
  style_set          = {},
  colored_set        = {},
  already            = false,
}
⟨/lua⟩
```

File II
`inline-helper.py` implementation

The standard header is managed specially because of the way docstrip automatically adds some header when extracting stuff from an archive. The next two lines are added by docstrip at the top of the preamble.

```
⟨*pyx⟩
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
⟨/pyx⟩
```

1 Header and global declarations

```
⟨*py⟩
__version__ = '0.10'
__YEAR__  = '2022'
__docformat__ = 'restructuredtext'

from posixpath import split
import sys
import argparse
import re
from pathlib import Path
from io import StringIO
import hashlib
import json
import pygments as P
import pygments.formatters.latex as L
from pygments.token import Token as PyToken
```

2 NLNLatexFormatter class

Based on pygments version 2.x. Enhanced formatter.

```
class NLNLatexFormatter(L.LatexFormatter):
  name = 'NLNLaTeX'
  aliases = []
  def __init__(self, *args, **kvargs):
    super().__init__(self, *args, **kvargs)
    self.escapeinside = kvargs.get('escapeinside', '')
    if len(self.escapeinside) == 2:
      self.left = self.escapeinside[0]
      self.right = self.escapeinside[1]
    else:
      self.escapeinside = ''
  def format_unencoded(self, tokensource, outfile):
    # TODO: add support for background colors
    t2n = self.ttype2name
    cp = self.commandprefix
    if self.full:
      realoutfile = outfile
      outfile = StringIO()
    outfile.write(r'\begin{Verbatim}[commandchars=\\\{\}')
    if self.linenos:
      start, step = self.linenostart, self.linenostep
      outfile.write(',numbers=left' +
            (start and ',firstnumber=%d' % start or '') +
            (step and ',stepnumber=%d' % step or ''))
    if self.mathescape or self.texcomments or self.escapeinside:
      outfile.write(r',codes={\catcode`\$=3\catcode`\^=7\catcode`\_=8}')
    if self.verboptions:
      outfile.write(',' + self.verboptions)
    outfile.write(']\n')
    for ttype, value in tokensource:
      if ttype in PyToken.Comment:
        if self.texcomments:
          # Try to guess comment starting lexeme and escape it ...
          start = value[0:1]
          for i in range(1, len(value)):
            if start[0] != value[i]:
              break
            start += value[i]

          value = value[len(start):]
          start = L.escape_tex(start, self.commandprefix)

          # ... but do not escape inside comment.
          value = start + value
        elif self.mathescape:
          # Only escape parts not inside a math environment.
          parts = value.split('$')
          in_math = False
          for i, part in enumerate(parts):
            if not in_math:
              parts[i] = L.escape_tex(part, self.commandprefix)
            in_math = not in_math
          value = '$'.join(parts)
        elif self.escapeinside:
          text = value
          value = ''
          while len(text) > 0:
            a,sep1,text = text.partition(self.left)
            if len(sep1) > 0:
              b,sep2,text = text.partition(self.right)
              if len(sep2) > 0:
                value += L.escape_tex(a, self.commandprefix) + b
              else:
                value += L.escape_tex(a + sep1 + b, self.commandprefix)
            else:
              value = value + L.escape_tex(a, self.commandprefix)
        else:
          value = L.escape_tex(value, self.commandprefix)
      elif ttype not in PyToken.Escape:
        value = L.escape_tex(value, self.commandprefix)
      styles = []
      while ttype is not PyToken:
        try:
          styles.append(t2n[ttype])
        except KeyError:
          # not in current style
          styles.append(L._get_ttype_name(ttype))
        ttype = ttype.parent
      styleval = '+'.join(reversed(styles))
      if styleval:
        spl = value.split('\n')
        for line in spl[:-1]:
          if line:
            outfile.write("\\%s{%s}{%s}" % (cp, styleval, line))
          outfile.write('\n')
        if spl[-1]:
          outfile.write("\\%s{%s}{%s}" % (cp, styleval, spl[-1]))
      else:
        outfile.write(value)

    outfile.write('\\end{Verbatim}\n')

    if self.full:
      realoutfile.write(DOC_TEMPLATE % dict(
        docclass  = self.docclass,
        preamble  = self.preamble,
        title     = self.title,
        encoding  = self.encoding or 'latin1',
        style_defs = self.get_style_defs(),
        code      = outfile.getvalue()
      ) )
```

3 Lexer class

`Lexer` This lexer takes one other lexer as argument, the lexer for the language being formatted, and the left and right delimiters for escaped text.

First everything is scanned using the language lexer to obtain strings and comments. All other consecutive tokens are merged and the resulting text is scanned for escaped segments, which are given the PyToken.Escape type. Finally text that is not escaped is scanned again with the language lexer.

```
class Lexer(P.lexer.Lexer):

  def __init__(self, left, right, lang, *args, **kvargs):
    self.left = left
    self.right = right
    self.lang = lang
    super().__init__(self, *args, **kvargs)

  def get_tokens_unprocessed(self, text):
    buf = ''
    for i, t, v in self.lang.get_tokens_unprocessed(text):
      if t in P.token.Comment or t in P.token.String:
        if buf:
          for x in self.get_tokens_aux(idx, buf):
            yield x
          buf = ''
        yield i, t, v
      else:
        if not buf:
          idx = i
        buf += v
    if buf:
      for x in self.get_tokens_aux(idx, buf):
        yield x

  def get_tokens_aux(self, index, text):
    while text:
      a, sep1, text = text.partition(self.left)
      if a:
        for i, t, v in self.lang.get_tokens_unprocessed(a):
          yield index + i, t, v
          index += len(a)
      if sep1:
        b, sep2, text = text.partition(self.right)
        if sep2:
          yield index + len(sep1), P.token.Escape, b
          index += len(sep1) + len(b) + len(sep2)
        else:
          yield index, P.token.Error, sep1
          index += len(sep1)
          text = b
```

4 Controller main class

The first class variables are string formats. They are used to let `inline-helper.py` talk back to TEX through `inline-helper.lua`.

```
class Controller:
  STY_FORMAT = r'''%%
\NLN_put:nn {style/%(name)s}{%%
%(defs)s%%
}%%
'''
  TEX_CALLBACK_FORMAT = r'''%%
\NLN_remove:n {colored:}%%
\NLN_style:nn {\tl_to_str:n {%(sty_p)s}}{\tl_to_str:n{%(name)s}}%%
\input {\tl_to_str:n {%(out_p)s}}%%
\NLN:n {colored:}%%
'''
  LUA_CALLBACK_FORMAT = r'''
NLN:cache_record(%(style)s),%(digest)s)
'''
  SNIPPET_FORMAT = r'''%%
\NLN_put:nn {colored} {%%
\group_begin:
\NLN:n {linenos:n} {%(line_numbers)s}%%
\begin{NLN/colored/%(mode)s/%(method)s}%%
%(body)s%%
\end{NLN/colored/%(mode)s/%(method)s}%%
\group_end:
}
'''
  PREAMBLE = r'''% -*- mode: latex -*-
\makeatletter
'''
  POSTAMBLE = r'''\makeatother
'''
```

4.1 Object nested class

```
  class Object(object):
    def __new__(cls, d={}, *args, **kvargs):
      if d.get('__cls__', 'arguments') == 'options':
        return super(Controller.Object, cls).__new__(
          Controller.Options, *args, **kvargs
        )
      else:
        return super(Controller.Object, cls).__new__(
          Controller.Arguments, *args, **kvargs
        )
    def __init__(self, d={}):
      for k, v in d.items():
        if type(v) == str:
          if v.lower() == 'true':
```

```
          setattr(self, k, True)
          continue
        elif v.lower() == 'false':
          setattr(self, k, False)
          continue
      setattr(self, k, v)
  def __repr__(self):
    return f"{object.__repr__(self)}: {self.__dict__}"
```

4.2 Options nested class

```
class Options(Object):
  lang = "tex"
  escapeinside = ""
  gobble = 0
  tabsize = 4
  style = 'default'
  texcomments = False
  mathescape =  False
  linenos = False
  linenostart = 1
  linenostep = 1
  linenosep = '0pt'
  encoding = 'guess'
  def __init__(self, *args, **kvargs):
    super().__init__(self, *args, **kvargs)
    try:
      lexer = P.lexers.get_lexer_by_name(self.lang)
    except P.util.ClassNotFound as err:
      sys.stderr.write('Error: ')
      sys.stderr.write(str(err))
    formatter = self.formatter = NLNLatexFormatter()
    escapeinside = self.escapeinside
    if len(escapeinside) == 2:
      left = escapeinside[0]
      right = escapeinside[1]
      formatter.escapeinside = escapeinside
      formatter.left = left
      formatter.right = right
      self.lexer = Lexer(left, right, lexer)
    gobble = abs(int(self.gobble))
    if gobble:
      lexer.add_filter('gobble', n=gobble)
    tabsize = abs(int(self.tabsize))
    if tabsize:
      lexer.tabsize = tabsize
    lexer.encoding = ''
    formatter.texcomments = self.texcomments
    formatter.mathescape = self.mathescape
    self.style = formatter.style = P.styles.get_style_by_name(self.style or self.sty)
```

4.3 Arguments nested class

```
class Arguments(Object):
  cache = False
  debug = False
```

```
    code = ""
    json = ""
    options = None
    directory = ""
```

4.4 Computed properties

`self.json_p` The full path to the `json` file containing all the data used for the processing.

(*End definition for* `self.json_p`. *This variable is documented on page* **??**.)

```
  _json_p = None
  @property
  def json_p(self):
    p = self._json_p
    if p:
      return p
    else:
      p = self.arguments.json
      if p:
        p = Path(p).resolve()
    self._json_p = p
    return p
```

`self.directory_p` The full path to the directory containing the various output files related to pygment. When not given in the `json` file, this is the directory of this file. The directory is created if necessary.

(*End definition for* `self.directory_p`. *This variable is documented on page* **??**.)

```
  _directory_p = None
  @property
  def directory_p(self):
    p = self._directory_p
    if p:
      return p
    p = self.arguments.directory
    if p:
      p = Path(p)
    else:
      p = self.json_p
      if p:
        p = p.parent / p.stem
      else:
        p = Path('SHARED')
    if p:
      p = p.resolve().with_suffix(".pygd")
      p.mkdir(exist_ok=True)
    self._directory_p = p
    return p
```

`self.colored_p` The full path to the file where colored commands created by pygment should be stored.

(*End definition for* `self.colored_p`. *This variable is documented on page* **??**.)

```
  _colored_p = None
  @property
  def colored_p(self):
    p = self._colored_p
    if p:
      return p
    p = self.arguments.output
    if p:
      p = Path(p).resolve()
    else:
      p = self.json_p
      if p:
        p = p.with_suffix(".pyg.tex")
    self._colored_p = p
    return p
```

`self.sty_p` The full path to the style file with definition created by pygment.

(*End definition for* `self.sty_p`. *This variable is documented on page* ??.)

```
  @property
  def sty_p(self):
    return (self.directory_p / self.options.style).with_suffix(".pyg.sty")
```

`self.parser` The correctly set up argarse instance.

(*End definition for* `self.parser`. *This variable is documented on page* ??.)

```
  @property
  def parser(self):
    parser = argparse.ArgumentParser(
      prog=sys.argv[0],
      description='''
Writes to the output file a set of LaTeX macros describing
the syntax highlighting of the input file as given by pygments.
'''
    )
    parser.add_argument(
      "-v", "--version",
      help="Print the version and exit",
      action='version',
      version=f'inline-helper version {__version__},'
      ' (c) {__YEAR__} by Jérôme LAURENS.'
    )
    parser.add_argument(
      "--debug",
      default=None,
      help="display informations useful for debugging"
    )
    parser.add_argument(
      "json",
      metavar="json data file",
      help="""
file name with extension of information to specify which processing is required
"""
    )
    return parser

```

4.5 Static methods

Controller.tex_command
Controller.lua_command
Controller.lua_command_now

self.tex_command(⟨*asynchronous tex command*⟩)
self.lua_command(⟨*asynchronous lua command*⟩)
self.lua_command_now(⟨*synchronous lua command*⟩)

Wraps the given command between markers. It will be in the output of the `inline-helper.py`, further captured by `inline-helper.lua` and either forwarded to TEX ot executed synchronously.

```
@staticmethod
def tex_command(cmd):
  print(f'<<<<<?TEX:{cmd}>>>>>')
@staticmethod
def lua_command(cmd):
  print(f'<<<<<?LUA:{cmd}>>>>>')
@staticmethod
def lua_command_new(cmd):
  print(f'<<<<<!LUA:{cmd}>>>>>')
```

4.6 Methods

4.6.1 `__init__`

__init__

Constructor. Reads the command line arguments.

```
def __init__(self, argv = sys.argv):
  argv = argv[1:] if re.match(".*inline-helper\.py$", argv[0]) else argv
  ns = self.parser.parse_args(
    argv if len(argv) else ['-h']
  )
  with open(ns.json, 'r') as f:
    self.arguments = json.load(
      f,
      object_hook=Controller.Object
    )
  self.options = self.arguments.options
  print("INPUT", self.json_p)
  print("OUTPUT DIR", self.directory_p)
  print("OUTPUT", self.colored_p)
```

4.6.2 `get_tex_p`

get_tex_p

⟨*variable*⟩ = self.get_tex_p(⟨*digest string*⟩)

The full path of the file where the colored commands created by `pygment` are stored. The digest allow to uniquely identify the code initially colored such that caching is easier.

```
def get_tex_p(self, digest):
  return (self.directory_p / digest).with_suffix(".pyg.tex")
```

4.6.3 read_input

```
def read_input(self, filename, encoding):
  with open(filename, 'rb') as infp:
    code = infp.read()
  if not encoding or encoding == 'guess':
    code, encoding = P.util.guess_decode(code)
  else:
    code = code.decode(encoding)
  return code, encoding
```

4.6.4 process

self.process

`self.process()`

Main entry point.

```
def process(self):
  arguments = self.arguments
  if self.convert_code():
    print('Done')
    return 0
  try:
    with open(self.arguments.output, 'w') as outfile:
      try:
        code, encoding = self.read_input(self.arguments.input, "guess")
      except Exception as err:
        print('Error: cannot read input file: ', err, file=sys.stderr)
        return 1
      self.convert(code, outfile, encoding)
  except Exception as err:
    print('Error: cannot open output file: ', err, file=sys.stderr)
    return 1
  print("Done")
  return 0
```

4.6.5 pygmentize

self.pygmentize

⟨*code variable*⟩, ⟨*style variable*⟩ = `self.pygmentize(`⟨*code*⟩, ⟨*inline_delim*⟩`)`

Where the ⟨*code*⟩ is pygmentized.

```
def pygmentize(self, code, inline_delim=True):
  options = self.options
  formatter = options.formatter
  formatter._create_stylesheet()
  style_defs = formatter.get_style_defs() \
    .replace(r'\makeatletter', '') \
    .replace(r'\makeatother', '') \
    .replace('\n', '%\n')
  ans_style  = self.STY_FORMAT % dict(
    name = options.style,
    defs = style_defs,
  )
  ans_code = []
```

```
    m = re.match(
      r'\\begin\{Verbatim}(.*)\n([\s\S]*?)\n\\end\{Verbatim}(\s*)\Z',
      P.highlight(code, options.lexer, formatter)
    )
    if m:
      linenos = options.linenos
      linenostart = abs(int(options.linenostart))
      linenostep = abs(int(options.linenostep))
      lines0 = m.group(2).split('\n')
      numbers = []
      lines = []
      counter = linenostart
      for line in lines0:
        line = re.sub(r'^ ', r'\vphantom{Xy}~', line)
        line = re.sub(r' ', '~', line)
        if linenos:
          if (counter - linenostart) % linenostep == 0:
            line = rf'\NLN:n {{lineno:}}{{{counter}}}' + line
            numbers.append(str(counter))
          counter += 1
        lines.append(line)
      ans_code.append(self.SNIPPET_FORMAT % dict(
        mode         = 'inline' if inline_delim else 'display',
        method       = self.arguments.method or 'default',
        line_numbers = ','.join(numbers),
        body         = '\\newline\n'.join(lines)
      ) )
    ans_code = "".join(ans_code)
    ans_code = re.sub(
      r"\expandafter\def\csname\s*(.*?)\endcsname",
      r'\cs_new:cpn{\1}',
      ans_code,
      flags=re.M
    )
    ans_code = re.sub(
      r"\csname\s*(.*?)\endcsname",
      r'\use:c{\1}',
      ans_code,
      flags=re.M
    )
    return ans_style, ans_code
```

4.6.6 `convert_code`

`self.convert_code` `self.convert_code()`

Call `self.pygmentize` and save the resulting style definitions and pygmented code in their respective locations.

```
  def convert_code(self):
    code = self.arguments.code
    if not code:
      return False
    style, code = self.pygmentize(code,True)
    sty_p = self.sty_p
    if self.arguments.cache and sty_p.exists():
      print("Already available:", sty_p)
    else:
      with sty_p.open(mode='w',encoding='utf-8') as f:
        f.write(style)
    h = hashlib.md5(str(code).encode('utf-8'))
    out_p = self.get_tex_p(h.hexdigest())
    if self.arguments.cache and out_p.exists():
      print("Already available:", out_p)
    else:
      with out_p.open(mode='w',encoding='utf-8') as f:
        f.write(self.PREAMBLE)
        print(f'DEBUG:{self.options}')
        f.write(code)
        f.write(self.POSTAMBLE)
    self.tex_command( self.TEX_CALLBACK_FORMAT % dict(
      sty_p = sty_p,
      out_p = out_p,
      name  = self.style,
    ) )
    if sty_p.parent.stem != 'SHARED':
      self.lua_command_now( self.LUA_CALLBACK_FORMAT % dict(
        style  = sty_p.name,
        digest = out_p.name,
      ) )
    print("PREMATURE EXIT")
    exit(1)
```

4.7 Main entry

```
if __name__ == '__main__':
  try:
    ctrl = Controller()
    sys.exit(ctrl.process())
  except KeyboardInterrupt:
    sys.exit(1)
⟨/py⟩
```

File III
inline.sty implementation

```
⟨*sty⟩
\makeatletter
```

1 Cache management

```
\AddToHook { begindocument/before } {
  \IfFileExists{./\jobname.aux}{}{
   \directlua{NLN:cache_clean()}
  }
}
\AddToHook { enddocument/end } {
  \directlua{NLN:cache_clean_unused()}
}
```

2 Constants

`\c_NLN_comment_prop` One line comment marker per language.

```
\prop_const_from_keyval:Nn \c_NLN_comment_prop {
  tex=\c_percent_str,
  lua=--,
  python=\c_hash_str,
  c=//,
  c++=//,
  javascript=//,
}
```

(*End definition for* `\c_NLN_comment_prop`*. This variable is documented on page* ??.)

3 Global properties

`\g/NLN/code/`
`\g/NLN/code/<name>`

Tree storage for global generic code properties or named code properties. These are overriden locally in environments using key-value actions. `\l_NLN_code_name_tl` is used as ⟨*name*⟩.

```
\prop_new:c {g/NLN/code/}
```

(*End definition for* `\g/NLN/code/` *and* `\g/NLN/code/<name>`*. These variables are documented on page* ??.)

`\l_NLN_code_name_tl` Locally used as ⟨*name*⟩ in `\g/NLN/code/<name>/` `\g/NLN/int/<name>` and similar.

```
\tl_new:N \l_NLN_code_name_tl
```

(*End definition for* `\l_NLN_code_name_tl`*. This variable is documented on page* ??.)

3.1 Management

`\NLN:n`
`\NLN:nn`

`\NLN:n {⟨key⟩}`
`\NLN:nn {⟨name⟩} {⟨key⟩}`

```
\cs_new:Npn \NLN:n #1 {
  \prop_item:cn {g/NLN/code/} { #1 }
}
\cs_new:Npn \NLN:nn #1 #2 {
  \prop_item:cn {g/NLN/code/#1/} { #2 }
}
```

\NLN_if_in:n*TF* ⋆
\NLN_if_in:nn*TF* ⋆

`\NLN_if_in:nTF {⟨key⟩} {⟨true code⟩} {⟨false code⟩}`
`\NLN_if_in:nnTF {⟨name⟩} {⟨key⟩} {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when `\g/NLN/code/` prop's contains ⟨`key`⟩, ⟨*false code*⟩ otherwise. Execute ⟨*true code*⟩ when `\g/NLN/code/⟨name⟩/` prop's contains ⟨`key`⟩, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN_if_in:n { T, F, TF } {
  \prop_if_in:cnTF {g/NLN/code/} { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \NLN_if_in:nn { T, F, TF } {
  \prop_if_in:cnTF {g/NLN/code/#1/} { #2 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\NLN:n*TF* ⋆

`\NLN:nNTF {⟨key⟩} ⟨tl var⟩ {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when `\g/NLN/code/` prop's ⟨`key`⟩ is retrieved in ⟨*tl var*⟩, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN:nN { T, F, TF } {
  \prop_get:cnNTF {g/NLN/code/} { #1 } #2 {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\NLN_put:nn
\NLN_put:nV
\NLN_gput:nn
\NLN_gput:nV
\NLN_put:nnn
\NLN_put:nnV
\NLN_gput:nnn
\NLN_gput:nnV

`\NLN_put:nn {⟨key⟩} {⟨value⟩}`
`\NLN_gput:nn {⟨key⟩} {⟨value⟩}`
`\NLN_put:nnn {⟨name⟩} {⟨key⟩} {⟨value⟩}`
`\NLN_gput:nnn {⟨name⟩} {⟨key⟩} {⟨value⟩}`

⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_put:nn #1 #2 {
  \prop_put:cnn {g/NLN/code/} { #1 } { #2 }
}
\cs_new:Npn \NLN_gput:nn #1 #2 {
  \prop_gput:cnn {g/NLN/code/} { #1 } { #2 }
}
\cs_generate_variant:Nn \NLN_put:nn { nV }
\cs_generate_variant:Nn \NLN_gput:nn { nV }
\cs_new:Npn \NLN_put:nnn #1 #2 #3 {
  \prop_put:cnn {g/NLN/code/#1/} { #2 } { #3 }
}
```

```
\cs_new:Npn \NLN_gput:nnn #1 #2 #3 {
  \prop_gput:cnn {g/NLN/code/#1/} { #2 } { #3 }
}
\cs_generate_variant:Nn \NLN_put:nnn { nnV }
\cs_generate_variant:Nn \NLN_gput:nnn { nnV }
```

\NLN_remove:n
\NLN_gremove:n
\NLN_remove:nn
\NLN_gremove:nn

\NLN_remove:n {⟨*value*⟩}
\NLN_remove:nn {⟨*key*⟩} {⟨*value*⟩}
\NLN_gremove:n {⟨*value*⟩}
\NLN_gremove:nn {⟨*key*⟩} {⟨*value*⟩}

```
\cs_new:Npn \NLN_remove:n #1 {
  \prop_remove:cn {g/NLN/code/} { #1 }
}
\cs_new:Npn \NLN_remove:nn #1 #2 {
  \prop_remove:cn {g/NLN/code/#1/} { #2 }
}
\cs_new:Npn \NLN_gremove:n #1 {
  \prop_gremove:cn {g/NLN/code/} { #1 }
}
\cs_new:Npn \NLN_gremove:nn #1 #2 {
  \prop_gremove:cn {g/NLN/code/#1/} { #2 }
}
```

3.2 Known keys and conditionals

\NLN_new_conditional:n

\NLN_new_conditional:n {⟨*key*⟩}

Create new conditionals for the given key. Does nothing out of this package..

```
\cs_new:Npn \NLN_new_conditional:n #1 {
  \exp_last_unbraced:Nx
  \prg_new_conditional:Nnn { \use:c {NLN_if_#1:} } { T, F, TF } {
    \group_begin:
    \NLN:nNTF { #1 } \l_tmpa_tl {
      \exp_args:NnV
      \regex_match:nnTF { ^\s*[tTyY] } \l_tmpa_tl
      { \group_end: \prg_return_true:  }
      { \group_end: \prg_return_false: }
    } { \group_end: \prg_return_false: }
  }
}
```

`format/code` Font/size/color specifier for inline code.

```
     \NLN_gput:nn { format/code } {
       \ttfamily
     }
```

`format/name` Font/size/color specifier for chunk name.

```
     \NLN_gput:nn { format/name } {
       \sffamily
```

```
      \scriptsize
      \color{gray}
    }
```

`format/lineno` Font/size/color specifier for line numbers.

```
    \NLN_gput:nn { format/lineno } {
      \sffamily
      \tiny
      \color{gray}
    }
```

`lang` the langage, defaults to `tex`

```
    \NLN_gput:nn { lang } { tex }
```

`lineno` show line numbers, defaults to `true`

```
    \NLN_gput:nn { show_lineno } { T }
```

`\NLN_if_show_lineno:`*`TF`* ⋆

`\NLN_if_show_lineno:TF {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

Execute ⟨*true code*⟩ when code property `show_lineo` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { show_lineno }
```

`name` show chunk names, defaults to `true`

```
    \NLN_gput:nn { show_name } { T }
```

`\NLN_if_show_name:`*`TF`* ⋆

`\NLN_if_show_name:TF {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

Execute ⟨*true code*⟩ when code property `show_name` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { show_name }
```

`only top` show names only on top, defaults to `true`

```
    \NLN_gput:nn { only_top } { T }
```

`\NLN_if_only_top:`*`TF`* ⋆

`\NLN_if_only_top:TF {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

Execute ⟨*true code*⟩ when code property `only_top` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { only_top }
```

`margin` use the margin to display line numbers and chunk names, defaults to `true`

```
    \NLN_gput:nn { use_margin } { T }
```

`\NLN_if_use_margin:TF` ⋆ `\NLN_if_use_margin:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code property `use_margin` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { use_margin }
```

`ignore` ignore that chunk or that export, defaults to `false`

```
    \NLN_gput:nn { ignore } { F }
```

`\NLN_if_ignore:TF` ⋆ `\NLN_if_ignore:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code property `ignore` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { ignore }
```

`reset` reset line numbering, defaults to `false`

```
    \NLN_gput:nn { reset } { F }
```

`\NLN_if_reset:TF` ⋆ `\NLN_if_reset:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code property `reset` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { reset }
```

`export` whether the code should be exported, defaults to `true`

```
    \NLN_gput:nn { export } { T }
```

`\NLN_if_export:TF` ⋆ `\NLN_if_export:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code property `export` is truthy, ⟨*false code*⟩ otherwise.

```
    \NLN_new_conditional:n { export }
```

`parskip` the parskip used to separate lines of code

```
    \AddToHook { begindocument/end } {
      \NLN_if_in:nF { parskip } {
        \exp_args:Nnx
        \NLN_gput:nn { parskip } { \the\parskip }
      }
    }
```

`baselinestretch` the baselinestretch used to separate lines of code

```
    \AddToHook { begindocument/end } {
      \NLN_if_in:nF { baselinestretch } {
        \exp_args:NnV
        \NLN_gput:nn { baselinestretch } \baselinestretch
      }
    }
```

`sep` the separation between inline code blocks and surrounding text.

```
      \NLN_gput:nn { sep } { 4pt plus 2pt minus 2pt }
```

`code` the cumulated inline code

```
      \NLN_gput:nn { .code } {}
```

Clean memory.

```
\cs_undefine:N \NLN_new_conditional:n
```

4 Counters

`\NLN_int_new:nn`

`\NLN_int_new:n {⟨name⟩} {⟨value⟩}`

Create an integer after ⟨*name*⟩ and set it globally to ⟨*value*⟩. ⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_int_new:nn #1 #2 {
  \int_new:c {g/NLN/int/#1}
  \int_gset:cn {g/NLN/int/#1} { #2 }
}
```

`\NLN_int_set:nn`
`\NLN_int_gset:nn`

`\NLN_int_set:n {⟨name⟩} {⟨value⟩}`

Set the integer named after ⟨*name*⟩ to the ⟨*value*⟩. `\NLN_int_gset:n` makes a global change. ⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_int_set:nn #1 #2 {
  \int_set:cn {g/NLN/int/#1} { #2 }
}
\cs_new:Npn \NLN_int_gset:nn #1 #2 {
  \int_gset:cn {g/NLN/int/#1} { #2 }
}
```

`\NLN_int_add:nn`
`\NLN_int_gadd:nn`

`\NLN_int_add:n {⟨name⟩} {⟨value⟩}`

Add the ⟨*value*⟩ to the integer named after ⟨*name*⟩. `\NLN_int_gadd:n` makes a global change. ⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_int_add:nn #1 #2 {
  \int_add:cn {g/NLN/int/#1} { #2 }
}
\cs_new:Npn \NLN_int_gadd:nn #1 #2 {
  \int_gadd:cn {g/NLN/int/#1} { #2 }
}
```

`\NLN_int_sub:nn`
`\NLN_int_gsub:nn`

`\NLN_int_sub:n {⟨name⟩} {⟨value⟩}`

Substract the ⟨*value*⟩ from the integer named after ⟨*name*⟩. `\NLN_int_gsub:n` makes a global change. ⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_int_sub:nn #1 #2 {
  \int_sub:cn {g/NLN/int/#1} { #2 }
}
\cs_new:Npn \NLN_int_gsub:nn #1 #2 {
  \int_gsub:cn {g/NLN/int/#1} { #2 }
}
```

`\NLN_int_if_exist:nTF`

`\NLN_int_if_exist:nTF {⟨name⟩} {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when an integer named after ⟨*name*⟩ exist, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN_int_if_exist:n { T, F, TF } {
  \int_if_exist:cTF {g/NLN/int/#1} {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\g/NLN/int/`
`\g/NLN/int/<name>`

Generic and named line number counter. `\l_NLN_code_name_t` is used as ⟨*name*⟩.

```
\NLN_int_new:nn {} { 1 }
```

(*End definition for* `\g/NLN/int/` *and* `\g/NLN/int/<name>`. *These variables are documented on page* **??**.)

`\NLN_int_use:n` ⋆

`\NLN_int_use:n {⟨name⟩}`

⟨*name*⟩ is a code name.

```
\cs_new:Npn \NLN_int_use:n #1 {
  \int_use:c {g/NLN/int/#1}
}
```

5 Variables

Line number counter for the code chunks.

`\g_NLN_code_int` Chunk number counter.

```
\int_new:N \g_NLN_code_int
```

(*End definition for* `\g_NLN_code_int`. *This variable is documented on page* **??**.)

`\g_NLN_code_prop` Global code property list.

```
\prop_new:N \g_NLN_code_prop
```

(*End definition for* `\g_NLN_code_prop`. *This variable is documented on page* **??**.)

`\g_NLN_export_prop` Global storage for ⟨*file name*⟩=⟨*comma separated chunk name*⟩

```
\prop_new:N \g_NLN_export_prop
```

(*End definition for* `\g_NLN_export_prop`. *This variable is documented on page* **??**.)

\l_NLN_prop Local scratch variable.

```
\prop_new:N \l_NLN_prop
```

(*End definition for* `\l_NLN_prop`*. This variable is documented on page* ??.)

\g_NLN_chunks_tl
\l_NLN_chunks_tl The comma separated list of current chunks. If the next list of chunks is the same as the current one, then it might not display.

```
\tl_new:N \g_NLN_chunks_tl
\tl_new:N \l_NLN_chunks_tl
```

(*End definition for* `\g_NLN_chunks_tl` *and* `\l_NLN_chunks_tl`*. These variables are documented on page* ??.)

\g_NLN_vars Tree storage for global variables.

```
\prop_new:N \g_NLN_vars
```

WHAT

(*End definition for* `\g_NLN_vars`*. This variable is documented on page* ??.)

\g_NLN_vars Tree storage for global variables.

```
\tl_new:N \g_NLN_hook_tl
```

(*End definition for* `\g_NLN_vars`*. This variable is documented on page* ??.)

\g/NLN/Chunks/<name> List of chunk keys for given named code.

(*End definition for* `\g/NLN/Chunks/<name>`*. This variable is documented on page* ??.)

5.1 Local variables

\l_NLN_recorded_tl Full verbatim body of the Inline environment.

```
\tl_new:N \l_NLN_recorded_tl
```

(*End definition for* `\l_NLN_recorded_tl`*. This variable is documented on page* ??.)

\g_NLN_int Global integer to store linenos locally in time.

```
\int_new:N \g_NLN_int
```

(*End definition for* `\g_NLN_int`*. This variable is documented on page* ??.)

\l_NLN_line_tl Token list for one line.

```
\tl_new:N \l_NLN_line_tl
```

(*End definition for* `\l_NLN_line_tl`*. This variable is documented on page* ??.)

\l_NLN_lineno_tl Token list for lineno display.

```
\tl_new:N \l_NLN_lineno_tl
```

(*End definition for* `\l_NLN_lineno_tl`*. This variable is documented on page* ??.)

\l_NLN_name_tl Token list for chunk name display.

```
\tl_new:N \l_NLN_name_tl
```

(*End definition for* `\l_NLN_name_tl`*. This variable is documented on page* ??.)

`\l_NLN_info_tl` Token list for the info of line.

```
\tl_new:N \l_NLN_info_tl
```

(*End definition for* `\l_NLN_info_tl`*. This variable is documented on page* **??**.)

`\l_NLN_clist` The comma separated list of current chunks.

```
\clist_new:N \l_NLN_clist
```

(*End definition for* `\l_NLN_clist`*. This variable is documented on page* **??**.)

`\l_NLN_in` Input file identifier

```
\ior_new:N \l_NLN_in
```

(*End definition for* `\l_NLN_in`*. This variable is documented on page* **??**.)

`\l_NLN_out` Output file identifier

```
\iow_new:N \l_NLN_out
```

(*End definition for* `\l_NLN_out`*. This variable is documented on page* **??**.)

6 Utilities

Utilities

`\NLN_clist_map_inline:Nnn`

`\NLN_clist_map_inline:Nnn` ⟨*clist var*⟩ `{`⟨ ⟩`}non empty code {`⟨ ⟩`}empty code`

Call `\clist_map_inline:Nnn` ⟨*clist var*⟩ `{`⟨*non empty code*⟩`}` when the list is not empty, execute metaempty code otherwise.

```
\cs_new:Npn \NLN_clist_map_inline:Nnn #1 #2 #3 {
  \clist_if_empty:NTF #1 { #3 } {
    \clist_map_inline:Nn #1 { #2 }
  }
}
```

`\NLN_process_record:` Record the current line or not.

```
\cs_new:Npn \NLN_process_record: {}
```

7 Shared key-value controls

Each action is meant to store the values in a code property, for the almost eponym key.

```
\keys_define:nn { NLN } {
```

Keys are:

`lineno[=true/false]` to display the line numbers, or not,

```
      lineno .code:n = \NLN_put:nn { show_lineno } { #1 },
      lineno .default:n = true,
```

name[=true/false] to display the chunk names

```
        name .code:n = \NLN_put:nn { show_name } { #1 },
        name .default:n = true,
```

only top to avoid chunk names repetitions, if on the same page, two consecutive code chunks have the same chunk names, the second names are not displayed.

```
        only~top .code:n = \NLN_put:nn { only_top } { #1 },
        only~top .default:n = true,
```

ignore to ignore chunks.

```
        ignore .code:n = \NLN_put:nn { ignore } { #1 },
        ignore .default:n = true,
```

margin[=true/false] to use the magin to display line numbers, or not,

```
        margin .code:n = \NLN_put:nn { use_margin } { #1 },
        margin .default:n = true,
```

lang=⟨*language name*⟩ , where ⟨*language name*⟩ is recognized by `pygment`,

```
        lang .code:n = \NLN_put:nn { lang } { #1 },
```

code format=⟨*format*⟩ , where ⟨*format*⟩ is used to display the code (mainly font, size and color),

```
        code~format .code:n = \NLN_put:nn { format/code } { #1 },
```

lineno format=⟨*format*⟩ , where ⟨*format*⟩ is used to display the line numbers (mainly font, size and color),

```
        name~format .code:n = \NLN_put:nn { format/name } { #1 },
```

name format=⟨*format*⟩ , where ⟨*name format*⟩ is used to display the chunk names (mainly font, size and color),

```
        lineno~format .code:n = \NLN_put:nn { format/lineno } { #1 },
```

post processor the name of the pygment post processor,

```
        post~processor .code:n = \NLN_put:nn { post_processor } { #1 },
```

post processor args the arguments of the pygment post processor,

```
        post~processor~args .code:n = \NLN_put:nn { post_processor_args } { #1 },
```

sep the separation with the surrounding text,

```
        sep .code:n = \NLN_put:nn { sep } { #1 },
```

parskip the value of the `\parskip` in inline code blocks,

```
      parskip .code:n = \NLN_put:nn { parskip } { #1 },
```

test whether the chunk is a test,

```
      test .code:n = \NLN_put:nn { test } { #1 },
```

```
  unknown .code:n = \PackageWarning
    { inline }
    { Unknown~option~‘\l_keys_key_str’ },
}
```

8 \InlineSet

`\InlineSet`

`\InlineSet {⟨key[=value] list⟩}`

To set up the package. This is executed at least once at the end of the preamble. The unique mandatory argument of `\InlineSet` is a list of ⟨*key*⟩`[=`⟨*value*⟩`]` items defined by

8.1 NLN/set key-value controls.

```
\keys_define:nn { } { NLN/set .inherit:n = NLN }
\keys_define:nn { NLN/set } {
```

minted to activate syntax coloring with `pygment`, calls `\_NLN_minted_on:` and forwards the argument as `minted` option,

```
      minted .code:n = {
        \_NLN_minted_on:
        \setkeys { minted@opt@g } { #1 }
      },
```

minted style=⟨*name*⟩ to select a predefined minted style, forwarded to `\usemintedstyle`,

```
      minted~style .code:n = {
        \RemoveFromHook { begindocument/before } [NLN/Minted]
        \AddToHook { begindocument/before } [NLN/Minted] {
          \usemintedstyle { #1 }
        }
      },
```

only description to typeset only the description section and ignore the implementation section.

```
      only~description .code:n = \prop_put:Nnn \l_NLN_vars
        { only_description } { #1 },
```

```
  unknown .code:n = \PackageWarning
    { NLN/set }
    { Unknown~option~‘\l_keys_key_str’ },
}
```

8.2 Implementation

```
\NewDocumentCommand \InlineSet { m } {
  \keys_set:nn { NLN/set } {#1}
  \NLN_if_use_minted:F {
    \bool_if:NT \g_NLN_minted_on_bool {
      \sys_if_shell:TF {
        \_NLN_if_pygmentize:TF {
          \bool_gset_true:N \g_NLN_use_minted_bool
        } {
          \msg_warning:nnn
            { inline }
            { :n }
            { No~"pygmentize"~found. }
        }
      } {
        \msg_warning:nnn
          { inline }
          { :n }
          { No~unrestricted~shell~escape~for~"pygmentize".}
      }
    }
  }
}
```

9 `InlineSplit` environment

10 `Inline` environment

`Inline` `\begin{⟨Inline⟩}{⟨key[=value] list⟩} ... \end{⟨Inline⟩}`

The ⟨*key*⟩[=⟨*value*⟩] items are defined by the

10.1 `NLN/code` key-value controls

```
\keys_define:nn { } { NLN/code .inherit:n = NLN }
\keys_define:nn { NLN/code } {
```

chunks=⟨*comma separated list of chunk names*⟩ When declaring an exported file, this is the list of chunks that will appear in that file. When declaring a code chunk, this the list of chunks where it will be stored. Chunks are collected unordered and ordered for comparison.

```
      chunks .clist_set:N = \l_NLN_clist,
```

reset[=<boolean string> When declaring an exported file, this is the list of chunks that will appear in that file. When declaring a code chunk, this the list of chunks where it will be stored. Chunks are collected unordered and ordered for comparison.

```
      reset .code:n = \NLN_put:nn { reset } { #1 },
      reset .default:n = true,
```

```
  unknown .code:n = \PackageWarning
    { NLN/code }
    { Unknown~option~'\l_keys_key_str' },
}
```

10.2 Implementation

`\NLN_if_record:TF`

`\NLN_if_record:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code should be recorded, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN_if_record: { T, F, TF } {
  \NLN_if_export:TF {
    \prg_return_true:
  } {
    \NLN_if_use_minted:TF {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  }
}
```

```
\cs_set:Npn \NLN_process_record: {
  \tl_put_right:Nx \l_NLN_recorded_tl { \the\verbatim@line \iow_newline: }
  \group_begin:
  \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
  \exp_args:Nx \directlua {NLN.records.append([===[\l_tmpa_tl]===])}
  \group_end:
}
```

```
\DeclareDocumentEnvironment { Inline } { m } {
  \directlua{NLN:start_recording()}
  \clist_clear:N \l_NLN_clist
  \keys_set:nn { NLN/code } { #1 }
  \clist_map_inline:Nn \l_NLN_clist {
    \NLN_int_if_exist:nF { ##1 } {
      \NLN_int_new:nn { ##1 } { 1 }
      \seq_new:c { g/NLN/chunks/##1 }
    }
  }
  \NLN_if_reset:T {
    \NLN_clist_map_inline:Nnn \l_NLN_clist {
      \NLN_int_gset:nn { ##1 } 1
    } {
      \NLN_int_gset:nn { } 1
    }
  }
  \tl_clear:N \l_NLN_code_name_tl
  \clist_map_inline:Nn \l_NLN_clist {
    \prop_concat:ccc
      {g/NLN/code/}
      {g/NLN/code/##1/}
      {g/NLN/code/}
    \tl_set:Nn \l_NLN_code_name_tl { ##1 }
    \clist_map_break:
  }
  \int_gset:Nn \g_NLN_int
    { \NLN_int_use:n { \l_NLN_code_name_tl } }
  \tl_clear:N \l_NLN_info_tl
```

```
\tl_clear:N \l_NLN_name_tl
\tl_clear:N \l_NLN_recorded_tl
\tl_clear:N \l_NLN_chunks_tl
\cs_set:Npn \verbatim@processline {
  \NLN_process_record:
}
\NLN_if_show_code:TF {
  \exp_args:NNx
  \skip_set:Nn \parskip { \NLN:n { parskip } }
  \clist_if_empty:NTF \l_NLN_clist {
    \tl_gclear:N \g_NLN_chunks_tl
  } {
    \clist_set_eq:NN \l_tmpa_clist \l_NLN_clist
    \clist_sort:Nn \l_tmpa_clist {
      \str_compare:nNnTF { ##1 } > { ##2 } {
        \sort_return_swapped:
      } {
        \sort_return_same:
      }
    }
    \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
    \NLN_if_show_name:T {
      \NLN_if_use_margin:T {
        \NLN_if_only_top:T {
          \tl_if_eq:NNT \l_tmpa_tl \g_NLN_chunks_tl {
            \tl_gset_eq:NN \g_NLN_chunks_tl \l_tmpa_tl
            \tl_clear:N \l_tmpa_tl
          }
        }
        \tl_if_empty:NF \l_tmpa_tl {
          \tl_set:Nx \l_NLN_chunks_tl {
            \clist_use:Nn \l_NLN_clist ,
          }
          \tl_set:Nn \l_NLN_name_tl {
            {
              \NLN:n { format/name }
              \l_NLN_chunks_tl :
              \hspace*{1ex}
            }
          }
        }
      }
      \tl_if_empty:NF \l_tmpa_tl {
        \tl_gset_eq:NN \g_NLN_chunks_tl \l_tmpa_tl
      }
    }
  }
  \if_mode_vertical:
  \else:
  \par
  \fi:
  \vspace{ \NLN:n { sep } }
  \noindent
  \frenchspacing
```

```
    \@vobeyspaces
    \normalfont\ttfamily
    \NLN:n { format/code }
    \hyphenchar\font\m@ne
    \@noligs
    \NLN_if_record:F {
      \cs_set_eq:NN \NLN_process_record: \prg_do_nothing:
    }
    \NLN_if_use_minted:F {
      \NLN_if_show_lineno:T {
        \NLN_if_use_margin:TF {
          \tl_set:Nn \l_NLN_info_tl {
            \hbox_overlap_left:n {
              {
                \l_NLN_name_tl
                \NLN:n { format/name }
                \NLN:n { format/lineno }
                \int_use:N \g_NLN_int
                \int_gincr:N \g_NLN_int
              }
              \hspace*{1ex}
            }
          }
        } {
          \tl_set:Nn \l_NLN_info_tl {
            {
              \NLN:n { format/name }
              \NLN:n { format/lineno }
              \hspace*{3ex}
              \hbox_overlap_left:n {
                \int_use:N \g_NLN_int
                \int_gincr:N \g_NLN_int
              }
            }
            \hspace*{1ex}
          }
        }
      }
      \cs_set:Npn \verbatim@processline {
        \NLN_process_record:
        \hspace*{\dimexpr \linewidth-\columnwidth}%
        \hbox_to_wd:nn { \columnwidth } {
          \l_NLN_info_tl
          \the\verbatim@line
          \color{lightgray}\dotfill
        }
        \tl_clear:N \l_NLN_name_tl
        \par\noindent
      }
    }
  } {
    \@bsphack
  }
  \group_begin:
```

```
  \g_NLN_hook_tl
  \let \do \@makeother
  \dospecials \catcode `\^^M \active
  \verbatim@start
} {
  \int_gsub:Nn \g_NLN_int {
    \NLN_int_use:n { \l_NLN_code_name_tl }
  }
  \int_compare:nNnT { \g_NLN_int } > { 0 } {
    \NLN_clist_map_inline:Nnn \l_NLN_clist {
      \NLN_int_gadd:nn { ##1 } { \g_NLN_int }
    } {
      \NLN_int_gadd:nn { } { \g_NLN_int }
    }
    \int_gincr:N \g_NLN_code_int
    \tl_set:Nx \l_tmpb_tl { \int_use:N \g_NLN_code_int }
    \clist_map_inline:Nn \l_NLN_clist {
      \seq_gput_right:cV { g/NLN/chunks/##1 } \l_tmpb_tl
    }
    \prop_gput:NVV \g_NLN_code_prop \l_tmpb_tl \l_NLN_recorded_tl
  }
  \group_end:
  \NLN_if_show_code:T {
  }
  \NLN_if_show_code:TF {
    \NLN_if_use_minted:TF {
      \tl_if_empty:NF \l_NLN_recorded_tl {
        \exp_args:Nnx \setkeys { FV } {
          firstnumber=\NLN_int_use:n { \l_NLN_code_name_tl },
        }
        \iow_open:Nn \minted@code { \jobname.pyg }
        \exp_args:NNV \iow_now:Nn \minted@code \l_NLN_recorded_tl
        \iow_close:N \minted@code
        \vspace* { \dimexpr -\topsep-\parskip }
        \tl_if_empty:NF \l_NLN_info_tl {
          \tl_use:N \l_NLN_info_tl
          \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
          \par\noindent
        }
        \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \NLN:n { lang } }
        %\DeleteFile { \jobname.pyg }
        \skip_vertical:n { -\topsep-\partopsep }
      }
    } {
      \exp_args:Nx \skip_vertical:n { \NLN:n { sep } }
      \noindent
    }
  } {
    \@esphack
  }
}
```

`NLN` \begin{⟨*NLN*⟩} ... \end{⟨*NLN*⟩}

Private environment.

```
\newenvironment{NLN}{
  \def \verbatim@processline {
    \group_begin:
    \NLN_processline_code_append:
    \group_end:
  }
%  \NLN_if_show_code:T {
%    \NLN_if_use_minted:TF {
%      \Needspace* { 2\baselineskip }
%    } {
%      \frenchspacing\@vobeyspaces
%    }
%  }
} {
  \NLN:nNTF { lang } \l_tmpa_tl {
    \tl_if_empty:NT \l_tmpa_tl {
      \clist_map_inline:Nn \l_NLN_clist {
        \NLN:nnNT { ##1 } { lang } \l_tmpa_tl {
          \tl_if_empty:NF \l_tmpa_tl {
            \clist_map_break:
          }
        }
      }
      \tl_if_empty:NT \l_tmpa_tl {
        \tl_set:Nn \l_tmpa_tl { tex }
      }
    }
  } {
    \tl_set:Nn \l_tmpa_tl { tex }
  }
  \clist_map_inline:Nn \l_NLN_clist {
    \NLN_gput:nnV { ##1 } { lang } \l_tmpa_tl
  }
}
```

`NLN.M` `\begin{⟨NLN.M⟩} ... \end{⟨NLN.N⟩}`

Private environment when `minted`.

```
\newenvironment{NLN_M}{
  \setkeys { FV } { firstnumber=last, }
  \clist_if_empty:NTF \l_NLN_clist {
    \exp_args:Nnx \setkeys { FV } {
      firstnumber=\NLN_int_use:n { },
  } } {
    \clist_map_inline:Nn \l_NLN_clist {
      \exp_args:Nnx \setkeys { FV } {
        firstnumber=\NLN_int_use:n { ##1 },
      }
      \clist_map_break:
  } }
  \iow_open:Nn \minted@code { \jobname.pyg }
  \tl_set:Nn \l_NLN_line_tl {
    \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
    \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
  }
} {
  \NLN_if_show_code:T {
    \NLN_if_use_minted:TF {
      \iow_close:N \minted@code
      \vspace* { \dimexpr -\topsep-\parskip }
      \tl_if_empty:NF \l_NLN_info_tl {
        \tl_use:N \l_NLN_info_tl
        \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
        \par\noindent
      }
      \exp_args:NV \minted@pygmentize \l_tmpa_tl
      \DeleteFile { \jobname.pyg }
      \vspace* { \dimexpr -\topsep -\partopsep }
    } {
      \@esphack
    }
  }
}
```

NLN.P `\begin{`⟨*NLN.P*⟩`} ... \end{`⟨*NLN.P*⟩`}`

Private pseudo environment. This is just a practical way of declaring balanced actions.

```
\newenvironment{NLN_P}{
  \if_mode_vertical:
    \noindent
  \else
    \vspace*{ \topsep }
    \par\noindent
  \fi
  \NLN_gset_chunks:
  \tl_if_empty:NTF \g_NLN_chunks_tl {
    \NLN_if_show_lineno:TF {
      \NLN_if_use_margin:TF {
```

No chunk name, line numbers in the margin

```
        \tl_set:Nn \l_NLN_info_tl {
          \hbox_overlap_left:n {
            \NLN:n { format/code }
            {
              \NLN:n { format/name }
              \NLN:n { format/lineno }
              \clist_if_empty:NTF \l_NLN_clist {
                \NLN_int_use:n { }
              } {
                \clist_map_inline:Nn \l_NLN_clist {
                  \NLN_int_use:n { ##1 }
                  \clist_map_break:
                }
              }
            }
            \hspace*{1ex}
          }
        }
      } {
```

No chunk name, line numbers not in the margin

```
        \tl_set:Nn \l_NLN_info_tl {
          {
            \NLN:n { format/code }
            {
              \NLN:n { format/name }
              \NLN:n { format/lineno }
              \hspace*{3ex}
              \hbox_overlap_left:n {
                \clist_if_empty:NTF \l_NLN_clist {
                  \NLN_int_use:n { }
                } {
                  \clist_map_inline:Nn \l_NLN_clist {
                    \NLN_int_use:n { ##1 }
                    \clist_map_break:
                  }
                }
              }
              \hspace*{1ex}
            }
          }
        }
      }
    } {
```

No chunk name, no line numbers

```
      \tl_clear:N \l_NLN_info_tl
    }
  } {
    \NLN_if_show_lineno:TF {
```

Chunk names, line numbers, in the margin

```
      \tl_set:Nn \l_NLN_info_tl {
        \hbox_overlap_left:n {
          \NLN:n { format/code }
          {
            \NLN:n { format/name }
            \g_NLN_chunks_tl :
            \hspace*{1ex}
            \NLN:n { format/lineno }
            \clist_map_inline:Nn \l_NLN_clist {
              \NLN_int_use:n { ####1 }
              \clist_map_break:
            }
          }
          \hspace*{1ex}
        }
        \tl_set:Nn \l_NLN_info_tl {
          \hbox_overlap_left:n {
            \NLN:n { format/code }
            {
              \NLN:n { format/name }
              \NLN:n { format/lineno }
              \clist_map_inline:Nn \l_NLN_clist {
                \NLN_int_use:n { ####1 }
                \clist_map_break:
              }
            }
            \hspace*{1ex}
          }
        }
      }
    } {
```

Chunk names, no line numbers, in the margin

```
      \tl_set:Nn \l_NLN_info_tl {
        \hbox_overlap_left:n {
          \NLN:n { format/code }
          {
            \NLN:n { format/name }
            \g_NLN_chunks_tl :
          }
          \hspace*{1ex}
        }
        \tl_clear:N \l_NLN_info_tl
      }
    }
  }
  \NLN_if_use_minted:F {
    \tl_set:Nn \l_NLN_line_tl {
      \noindent
      \hbox_to_wd:nn { \textwidth } {
        \tl_use:N \l_NLN_info_tl
        \NLN:n { format/code }
        \the\verbatim@line
        \hfill
      }
      \par
    }
    \@bsphack
  }
} {
  \vspace*{ \topsep }
  \par
  \@esphack
}
```

11 \InlineExport

\InlineExport   \InlineExport{⟨*key[=value] list*⟩}

The ⟨*key*⟩[=⟨*value*⟩] items are defined by

11.1 NLN/export key-value controls

```
\keys_define:nn { } { NLN/export .inherit:n = NLN/code }
\keys_define:nn { NLN/export } {
```

file the output file name

```
      file .tl_set:N = \l_NLN_tl,
      file .value_required:n = true,
```

preamble the added preamble.

```
      preamble .code:n = \prop_put:Nnn \l_NLN_vars { preamble } { #1 },
```

raw to remove any additional material,

```
      raw .code:n = \prop_put:Nnn \l_NLN_vars { raw } { #1 },
```

```
  unknown .code:n = \PackageWarning
    { NLN/export }
    { Unknown~option~`\l_keys_key_str' },
}
```

11.2 Implementation

```
\DeclareDocumentCommand \InlineExport { m } {
  \group_begin:
  \clist_clear:N \l_NLN_clist
  \prop_clear:c {g/NLN/code/}
  \prop_put:cnn {g/NLN/code/} { lang } { tex }
  \keys_set:nn { NLN/export } { #1 }
  \prop_gput:NVV \g_NLN_export_prop \l_NLN_tl \l_NLN_clist
  \prop_gput:cnV { g/NLN/export/\l_NLN_tl } { chunks } \l_NLN_clist
  \prop_gput:cnx { g/NLN/export/\l_NLN_tl } { preamble }
    { \prop_item:Nn \l_NLN_vars { preamble } }
  \bool_set:Nx \l_tmpa_bool { \prop_item:Nn \l_NLN_vars { raw } }
  \prop_gput:cnV { g/NLN/export/\l_NLN_tl } { preamble } \l_tmpa_bool
  \NLN:nNT { lang } \l_tmpa_tl {
    \clist_map_inline:Nn \l_NLN_clist {
      \prop_gconcat:ccc
        {g/NLN/code/##1/}
        {g/NLN/code/##1/}
        {g/NLN/code/}
    }
  }
  \group_end:
}
```

Files are created at the end of the typesetting process.

```
\AddToHook { enddocument / end } {
  \group_begin:
  \prop_map_inline:Nn \g_NLN_export_prop {
    \iow_open:Nn \l_NLN_out { #1 }
    \iow_term:x { Exporting~chunks~#2~to~#1 }
    \prop_get:cnNF { g/NLN/export/#1 } { raw } \l_tmpa_bool {
      \bool_set_false:N \l_tmpa_bool
    }
    \bool_if:NF \l_tmpa_bool {
      \prop_get:cnNT { g/NLN/export/#1 } { preamble } \l_tmpa_tl {
        \prop_get:cnNF { g/NLN/export/#1 } { lang } \l_tmpa_str {
          \str_set:Nn \l_tmpa_str { tex }
        }
        \prop_get:NVNTF \c_NLN_comment_prop \l_tmpa_str \l_tmpa_str {
          \tl_set:Nn \l_tmpb_tl {
            \l_tmpa_str\l_tmpa_str\space\space
          }
        } {
          \tl_clear:N \l_tmpb_tl
        }
        \tl_put_right:Nx \l_tmpb_tl {
          This~is~file~‘#1’~
          generated~from~‘\c_sys_jobname_str.tex’~on~\DTMnow.
        }
        \iow_now:Nx \l_NLN_out { \l_tmpb_tl }
        \iow_now:Nx \l_NLN_out { \l_tmpa_tl }
      }
    }
    \clist_map_inline:nn { #2 } {
      \NLN:nnNT { ##1 } { .code } \l_tmpa_tl {
        \exp_args:NNV \iow_now:Nn \l_NLN_out \l_tmpa_tl
      }
    }
    \iow_close:N \l_NLN_out
  }
  \group_end:
}
```

12 Management

`\g_NLN_in_impl_bool` Whether we are currently in the implementation section.

```
\bool_new:N \g_NLN_in_impl_bool
```

(*End definition for* `\g_NLN_in_impl_bool`*. This variable is documented on page* **??**.)

`\NLN_if_show_code:TF` `\NLN_if_show_code:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when code should be printed, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN_if_show_code: { T, F, TF } {
  \bool_if:nTF {
    \g_NLN_in_impl_bool && !\g_NLN_with_impl_bool
  } {
    \prg_return_false:
  } {
    \prg_return_true:
  }
}
```

`\g_NLN_with_impl_bool`

```
\bool_new:N \g_NLN_with_impl_bool
```

(*End definition for* `\g_NLN_with_impl_bool`. *This variable is documented on page* ??.)

13 All purpose messaging

14 `minted` and `pygment`

`\g_NLN_minted_on_bool` Whether minted is available, initially set to `false`.

```
\bool_new:N \g_NLN_minted_on_bool
```

(*End definition for* `\g_NLN_minted_on_bool`. *This variable is documented on page* ??.)

`\g_NLN_use_minted_bool` Whether minted is used, initially set to `false`.

```
\bool_new:N \g_NLN_use_minted_bool
```

(*End definition for* `\g_NLN_use_minted_bool`. *This variable is documented on page* ??.)

`\NLN_if_use_minted:TF`

`\NLN_if_use_minted:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when using `minted`, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \NLN_if_use_minted: { T, F, TF } {
  \bool_if:NTF \g_NLN_use_minted_bool
    { \prg_return_true:  }
    { \prg_return_false: }
}
```

`\_NLN_if_pygmentize:TF`

`\NLN_if_pygmentize:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when `pygmentize` is available, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn\_NLN_if_pygmentize: { T, F, TF } {
  \group_begin:
  \sys_get_shell:nnN {which~pygmentize} {} \l_tmpa_tl
  \tl_if_empty:NTF \l_tmpa_tl {
    \tl_set:Nn \l_tmpa_tl { \prg_return_false: }
  } {
    \tl_set:Nn \l_tmpa_tl { \prg_return_true: }
  }
  \exp_last_unbraced:NV
  \group_end: \l_tmpa_tl
}
```

`\_NLN_minted_on:`

`\_NLN_minted_on:`

Private function. During the preamble, loads `minted`, sets `\g_NLN_minted_on_bool` to `true` and prepares `pygment` processing.

```
\cs_set:Npn \_NLN_minted_on: {
  \bool_gset_true:N \g_NLN_minted_on_bool
  \RequirePackage{minted}
  \setkeys{ minted@opt@g } { linenos=false }
  \minted@def@opt{post~processor}
  \minted@def@opt{post~processor~args}
  \pretocmd\minted@inputpyg{
    \NLN@postprocesspyg {\minted@outputdir\minted@infile}
  }{}{\fail}
```

In the execution context of `\minted@inputpyg`,

`#1` is the name of the python script, e.g., “`process.py`”

`#2` is the input “.pygtex” file “`\minted@outputdir\minted@infile`”

`#3` are more args passed to the python script, possibly empty

```
  \newcommand{\NLN@postprocesspyg}[1]{%
    \group_begin:
    \tl_set:Nx \l_tmpa_tl {\NLN:n { post_processor } }
    \tl_if_empty:NF \l_tmpa_tl {
```

Execute ‘`python3 <script.py> <file.pygtex> <more_args>`‘

```
      \tl_set:Nx \l_tmpb_tl {\NLN:n { post_processor_args } }
      \exp_args:Nx
      \sys_shell_now:n {
        python3\space
        \l_tmpa_tl\space
        ##1\space
        \l_tmpb_tl
      }
    }
    \group_end:
  }
}
```

```
%\AddToHook { begindocument / end } {
%  \cs_set_eq:NN \_NLN_minted_on: \prg_do_nothing:
%}
```

Utilities to setup `pygment` post processing. The `pygment` post processor marks some code with `\InlineEmph`.

```
\ProvideDocumentCommand{\InlineEmph}{m}{\textcolor{red}{#1}}
```

`\InlineStorePreamble` `\InlineStorePreamble {⟨variable⟩} {⟨file name⟩}`

Store the content of ⟨*file name*⟩ into the variable ⟨*variable*⟩.

15 Separators

`\InlineImplementation` `\InlineImplementation`

Start an implementation part where all the sectioning commands do nothing.

`\InlineFinale` `\InlineFinale`

Stop an implementation part.

16 Finale

```
\DeclareDocumentCommand \InlineStorePreamble { m m } {
  \group_begin:
  \msg_info:nnn
    { inline }
    { :n }
    { Reading~preamble~from~file~"#2". }
  \tl_clear:N \g_tmpa_tl
  \tl_clear:N \g_tmpb_tl
  \ior_open:Nn \l_NLN_in { #2 }
  \bool_until_do:nn { \ior_if_eof_p:N \l_NLN_in } {
    \ior_str_get:NN \l_NLN_in \l_tmpa_tl
    \tl_if_empty:NTF \l_tmpa_tl {
      \tl_put_right:Nn \g_tmpb_tl { \iow_newline: }
    } {
      \tl_put_right:Nx \g_tmpa_tl { \g_tmpb_tl }
      \tl_set:Nn \g_tmpb_tl { \iow_newline: }
      \tl_put_right:NV \g_tmpa_tl \l_tmpa_tl
    }
  }
  \ior_close:N \l_NLN_in
  \exp_args:NNNx
  \group_end:
  \tl_set:Nn #1 { \tl_to_str:N \g_tmpa_tl }
}
\newcounter{NLN@impl@page}
\DeclareDocumentCommand \InlineImplementation {} {
  \bool_if:NF \g_NLN_with_impl_bool {
    \clearpage
    \bool_gset_true:N \g_NLN_in_impl_bool
    \let\NLN@old@part\part
    \DeclareDocumentCommand\part{som}{}
    \let\NLN@old@section\section
    \DeclareDocumentCommand\section{som}{}
    \let\NLN@old@subsection\subsection
    \DeclareDocumentCommand\subsection{som}{}
    \let\NLN@old@subsubsection\subsubsection
    \DeclareDocumentCommand\subsubsection{som}{}
    \let\NLN@old@paragraph\paragraph
    \DeclareDocumentCommand\paragraph{som}{}
    \let\NLN@old@subparagraph\subparagraph
    \DeclareDocumentCommand\subparagraph{som}{}
    \cs_if_exist:NT \refsection{ \refsection }
    \setcounter{ NLN@impl@page }{ \value{page} }
  }
}
\DeclareDocumentCommand\InlineFinale {} {
  \bool_if:NF \g_NLN_with_impl_bool {
    \clearpage
    \bool_gset_false:N \g_NLN_in_impl_bool
    \let\part\NLN@old@part
    \let\section\NLN@old@section
    \let\subsection\NLN@old@subsection
    \let\subsubsection\NLN@old@subsubsection
    \let\paragraph\NLN@old@paragraph
    \let\subparagraph\NLN@old@subparagraph
    \setcounter { page } { \value{ NLN@impl@page } }
  }
}
\cs_set_eq:NN \NLN_line_number: \prg_do_nothing:
```

17 Finale

```
\AddToHook { cmd/FancyVerbFormatLine/before } {
  \NLN_line_number:
}
\AddToHook { shipout/before } {
  \tl_gclear:N \g_NLN_chunks_tl
}
\InlineSet {}
```

```
% =========================================================
% Auxiliary:
%   finding the widest string in a comma
%   separated list of strings delimited by parenthesis
% =========================================================

% arguments:
% #1) text: a comma separeted list of strings
% #2) formatter: a macro to format each string
% #3) dimension: will hold the result

\cs_new:Npn \NLNWidest (#1) #2 #3 {
  \group_begin:
  \dim_set:Nn #3 { 0pt }
  \clist_map_inline:nn { #1 } {
    \hbox_set:Nn \l_tmpa_box { #2{##1} }
    \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
    \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
      \dim_set_eq:NN #3 \l_tmpa_dim
    }
  }
  \exp_args:NNNV
  \group_end:
  \dim_set_eq:NN #3 #3
}
\ExplSyntaxOff

```

18 **pygmentex** implementation

```
% =========================================================
% fancyvrb new commands to append to a file
% =========================================================

% See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-
chars-and-verbatim

```

```
\ExplSyntaxOn

\seq_new:N \l_NLN_records_seq

\long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}

\def\VerbatimOutAppend{\FV@Environment{}{VerbatimOutAppend}}

\def\FVB@VerbatimOutAppend#1{%
  \@bsphack
  \begingroup
    \seq_clear:N \l_NLN_records_seq
    \FV@UseKeyValues
    \FV@DefineWhiteSpace
    \def\FV@Space{\space}%
    \FV@DefineTabOut
    \def\FV@ProcessLine{%##1
%       \seq_put_right:Nn \l_NLN_records_seq { ##1 }%
      \immediate\unexpanded@write#1%{##1}
    }%
    \let\FV@FontScanPrep\relax
    \let\@noligs\relax
    \FV@Scan
}

\def\FVE@VerbatimOutAppend{
  \seq_use:Nn \l_NLN_records_seq /
  \endgroup
  \@esphack
}

\DefineVerbatimEnvironment{VerbatimOutAppend}{VerbatimOutAppend}{}
% ==========================================================
% Main options
% ==========================================================

\newif\ifNLN@left
\newif\ifNLN@right


```

19 Display technology

Inserting code snippets follows one of two modes: run or block. The former is displayed as running text and used by the `\NLNCode` command whereas the latter is displayed as a separate block and used by the `NLN/Code` environment. Both have one single required argument, which is a ⟨*key-value*⟩ configuration list named `NLN_code`. The contents is then colorized with the aid of `inline-helper.py` which will return some code enclosed within an environment created by one of `\NLNNewRunMethod`, `\NLNRenewRunMethod`, `\NLNNewBlockMethod`, `\NLNRenewBlockMethod` functions.

19.1 `\NLNCode` run function

Only the body of the NLN/Code environment may be exported.

`\NLNCode`

`\NLNCode{`⟨*configuration*⟩`}`⟨*delimiter*⟩⟨*code*⟩⟨*same delimiter*⟩

```
\NewDocumentCommand \NLNCode { mm } {
  \group_begin:
    \prop_concat:ccc {l_NLN_prop} {c_empty_prop} {g/NLN/prop} % NO \prop_set_eq:Nc
    \cs_set:Npn \NLN_put:nn ##1 ##2 {
      \prop_put:Nnn \l_NLN_prop { ##1 } { ##2 }% expand the value?
    }
    \keys_set:nn { NLN } { #1 }
    \directlua{NLN:options_reset()}
    \prop_map_inline:Nn \l_NLN_prop {
      \lua_now:e {NLN:option_add('\lua_escape:n {##1}', '\lua_escape:n {##2}')}
    }
    VERB:#2
    \DefineShortVerb{#2}%
    \SaveVerb
      [aftersave={%
       \UndefineShortVerb{#2}%
       \lua_now:e {NLN:process_run('FV@SV@NLN')}
       \group_end:
      }]%
      {NLN}#2%
}
```

19.2 `NLN/Code` environment

19.3 Creating display methods

`\NLNNewRunMethod`
`\NLNRenewRunMethod`
`\NLNNewBlockMethod`
`\NLNRenewBlockMethod`

`\NLNNewRunMethod{`⟨*method name*⟩`}{`⟨*method body*⟩`}`
`\NLNRenewRunMethod{`⟨*method name*⟩`}{`⟨*method body*⟩`}`
`\NLNNewBlockMethod{`⟨*method name*⟩`}{`⟨*begin instructions*⟩`}{`⟨*end instructions*⟩`}`
`\NLNRenewBlockMethod{`⟨*method name*⟩`}{`⟨*begin instructions*⟩`}{`⟨*end instructions*⟩`}`

{⟨*method name*⟩} is a non void string. The run methods create a command with a unique argument which the colored code. The block methods create an environent. The body of the environment is available in the `\NLNBody` variable. The options passed with the `options` key are available in the `\NLNOptions` variable.

```
\cs_new:Npn \NLNNewRunMethod #1 #2 {
  \cs_new:cpn { NLN/colored/run/#1: } ##1 {
    #2
  }
  \ignorespaces
}
\cs_new:Npn \NLNRenewRunMethod #1 #2 {
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
    { NLN/method }
    { The~method~cannot~be~void. }
  } {
    \cs_if_exist:cTF { NLN/colored/run/#1: } {
      \cs_set:cpn { NLN/colored/run/#1: } ##1 {
        #2
      }
    } {
      \PackageWarning
      { NLN/method }
      { No~run~method~#1.}
    }
  \ignorespaces
  }
}
\cs_new:Npn \NLNNewBlockMethod #1 #2 #3 {
  \NewDocumentEnvironment { NLN/colored/block/#1 } { +b } {
    \exp_args:NNx \tl_set:Nn \NLNOptions { \NLN:n { options } }
    \tl_set:Nn \NLNBody { #1 }
    #2
  } { #3 }
}
\cs_new:Npn \NLNRenewBlockMethod #1 #2 #3 {
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
    { NLN/method }
    { The~method~cannot~be~void. }
    \use_none:nn
  } {
    \RenewDocumentEnvironment { NLN/colored/block/#1 } { +b } {
      \exp_args:NNx \tl_set:Nn \NLNOptions { \NLN:n { options } }
      \def \NLNBody { #1 }
      #2
    } { #3 }
  }
}
```

19.4 Run mode default method

```
\NLNNewRunMethod {} {
} {
}
```

19.5 Run mode `efbox` method

`NLNCallWithOptions` ⋆

`\NLNCallWithOptions`⟨*cs*⟩

Call ⟨*cs*⟩, assuming it has a first optional argument. It will receive the arguments passed to `\NLNCode` with the `options` key.

```
\cs_new:Npn \NLNCallWithOptions #1 {
  \exp_last_unbraced:NNx
  #1[\NLN:n { options }]
}
\NLNNewRunMethod {efbox} {
  \NLNCallWithOptions\efbox{#1}%
}
```

19.6 Block mode default method

```
\NLNNewBlockMethod {} {
} {
}
```

19.7 key-value action

method=⟨*method name*⟩ , where ⟨*method name*⟩ is recognized by inline

```
\keys_define:nn { NLN/.method } {
  method .code:n = \NLN_put:nn { method } { #1 },
}
```

20 Shared key-value controls

These declare the interface of the various commands and environments.

20.1 inline key-value controls

Each action is meant to store the values in a code property, for the almost eponym key. The setter is `\NLN_put:nn` except for options which is `\NLN_option_put:nn`. These are defined just before reading the options. Keys are:

```
\tl_new:N \l_NLN_options_tl
\keys_define:nn { NLN } {
```

lang=⟨*language name*⟩ , where ⟨*language name*⟩ is recognized by `pygment`,

method=⟨*method name*⟩ , switcher for different methods,

lineno[=true/false] to display the line numbers, or not,

```
      lineno .code:n = \NLN_put:nn { show_lineno } { #1 },
      lineno .default:n = true,
```

name[=true/false] to display the chunk names

```
      name .code:n = \NLN_put:nn { show_name } { #1 },
      name .default:n = true,
```

only top to avoid chunk names repetitions, if on the same page, two consecutive code chunks have the same chunk names, the second names are not displayed.

```
      only~top .code:n = \NLN_put:nn { only_top } { #1 },
      only~top .default:n = true,
```

ignore to ignore chunks.

```
      ignore .code:n = \NLN_put:nn { ignore } { #1 },
      ignore .default:n = true,
```

margin[=true/false] to use the magin to display line numbers, or not,

```
      margin .code:n = \NLN_put:nn { use_margin } { #1 },
      margin .default:n = true,
```

`format=`⟨***`kv format items`***⟩ , where ⟨*kv format items*⟩ are detailed below,

`format/code=`⟨***`format`***⟩ , where ⟨*format*⟩ is used to display the code (mainly font, size and color),

`format/lineno=`⟨***`format`***⟩ , where ⟨*format*⟩ is used to display the line numbers (mainly font, size and color),

`name format=`⟨***`format`***⟩ , where ⟨*name format*⟩ is used to display the chunk names (mainly font, size and color),

```
      format .code:n = \keys_set:nn {NLN/format} { #1 },
```

`sep` the separation with the surrounding text,

`parskip` the value of the `\parskip` in inline code blocks,

`baselinestretch` the value of the `\baselinestretch` in inline code blocks,

`test` whether the chunk is a test,

```
      test .code:n = \NLN_put:nn { is_test } { #1 },
      test .default:n = true,
```

`anything` forwards to,

```
      unknown .code:n = {
        \group_begin:
        \exp_args:NnV
        \regex_extract_once:nnNTF { ^options/(.*) } \l_keys_key_str \l_tmpa_seq {
          \tl_set:Nx \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
          \tl_put_right:Nn \l_tmpa_tl { = #1 }
          \exp_args:NNnV
          \group_end:
          \keys_set:nn { NLN/options } \l_tmpa_tl
        } {
         \group_end:
        }
      },
```

```
}
```

20.2 `options` key-value controls

We accept any value because we do not know in advance the real target. Everything is collected in `\l_NLN_options_clist`.

`\l_NLN_options_clist` All the ⟨*key[=value] items*⟩ passed as options are collected here. This hould be cleared before arguments are parsed.

(*End definition for `\l_NLN_options_clist`. This variable is documented on page* **??**.)

```
\clist_new:N \l_NLN_options_clist
```

There are 2 ways to collect options:

```
\keys_define:nn { NLN/options } {
  unknown .code:n = {
    \group_begin:
    \tl_set_eq:NN \l_tmpa_tl \l_keys_key_str
    \tl_put_right:Nn \l_tmpa_tl { = #1 }
    \exp_args:NNNV
    \group_end:
    \clist_put_right:Nn \l_NLN_options_clist \l_tmpa_tl
  }
}
```

`options=`⟨***`options key value items`***⟩ , where ⟨*options key value items*⟩ are display options forwarded to other packages.

```
      options .code:n = {
        \clist_put_right:Nn \l_NLN_options_clist { #1 }
      },
```

21 Something else

some settings used by fancyvrb: * for line numbering: numbers, numbersep, firstnumber, stepnumber, numberblanklines * for selection of lines to print: firstline, lastline,

```
\pgfkeys{%
  /NLN/.cd,
  %
  %
  lang/.code           = \NLN_put:nn {lang} { #1 },
  sty/.code            = \NLN_put:nn {sty} { #1 },
  escapeinside/.code   = \NLN_put:nn {escapeinside} { #1 },
  texcomments/.code    = \NLN_put:nn {texcomments} { #1 },% boolean
  mathescape/.code     = \NLN_put:nn {mathescape} { #1 },% boolean
  %
  label/.code          = \NLN_put:nn {label} { #1 },
  caption/.code        = \NLN_put:nn {caption} { #1 },
  %
  gobble/.code         = \NLN_put:nn {gobble} { #1 },
  tabsize/.code        = \NLN_put:nn {tabsize} { #1 },
  %
  linenos/.code        = \NLN_put:nn {linenos} { #1 },% boolean
  linenostart/.code    = \NLN_put:nn {linenostart} { #1 },
  linenostep/.code     = \NLN_put:nn {linenostep} { #1 },
  linenosep/.code      = \NLN_put:nn {linenosep} { #1 },
  %
  colback/.code        = \NLN_put:nn {colback} { #1 },
  font/.code           = \NLN_put:nn {font} { #1 },
  %
  texcomments/.default = true,
  mathescape/.default  = true,
  linenos/.default     = true,
}

```

```
\pgfqkeys{/NLN}{
  boxing~method = mdframed,
  inline~method = efbox,
  sty           = default,
  linenos       = false,
  linenosep     = 2pt,
  font          = \ttfamily,
  tabsize       = 0,
}

% =========================================================
% pygmented commands and environments
% =========================================================

\newwrite\NLN@outfile

\newcount\NLN@counter

\newcommand\NLN@process@options[1]{%
  \pgfkeys{%
    /pgf/key~filters/defined/.install~key~filter,%
    /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingGlo
  }%
  \def\NLNRemainingGlobalOptions{}%
  \pgfkeysalsofilteredfrom{\NLN@global@options}%
  \pgfkeysalso{%
    /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingUse
  }%
  \def\NLNRemainingUserOptions{}%
  \pgfqkeysfiltered{/NLN}{#1}%
  % %%%%%% DEBUGING
  % \typeout{}%
  % \typeout{\string\NLN@global@options:}\typeout{\meaning\NLN@global@options}%
  % \typeout{\string\NLNRemainingGlobalOptions:}\typeout{\meaning\NLNRemainingGlobalOptions}
  % \typeout{\string\NLNRemainingUserOptions:}\typeout{\meaning\NLNRemainingUserOptions}%
  %
  \fvset{gobble=0,tabsize=0}%
}

\newcommand\NLN@process@more@options[1]{%
  \pgfkeysalso{%
    /pgf/key~filters/false/.install~key~filter,%
    /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingOpt
  }%
  \def\NLNRemainingOptions{}%
  \pgfkeysalsofilteredfrom{\NLNRemainingGlobalOptions}%
  \cs_if_exist:cT {NLN@#1@more@options} {
    \exp_args:Nx
    \pgfkeysalsofilteredfrom { \use:c{NLN@#1@more@options}, }
  }
  \pgfkeysalsofilteredfrom{\NLNRemainingUserOptions}%
  % %%%%%% DEBUGING
  % \typeout{}%
  % \typeout{\string\NLNRemainingOptions:}%
  % \typeout{\meaning\NLNRemainingOptions}%
}

\newcommand\inputpygmented[2][]{%
  \begingroup
    \NLN@process@options{#1}%
    \immediate\write\NLN@outfile{<@@NLN@input@\the\NLN@counter}%
    \immediate\write\NLN@outfile{\exp_args:NV\detokenize\NLN@global@options,\detokenize{#1}}
    \immediate\write\NLN@outfile{#2}%
    \immediate\write\NLN@outfile{>@@NLN@input@\the\NLN@counter}%
    %
    \csname NLN@snippet@\the\NLN@counter\endcsname
    \global\advance\NLN@counter by 1\relax
  \endgroup
}

\NewDocumentEnvironment{pygmented}{+O{}m}{%
  \lua_now:e {NLN:start_recording()}
  \NLN@process@options{#1}%
  \immediate\write\NLN@outfile{<@@NLN@display@\the\NLN@counter}%
  \immediate\write\NLN@outfile{
    \exp_args:NV\detokenize\NLN@global@options,\detokenize{#1}
  }%
  \VerbatimEnvironment
  \begin{VerbatimOutAppend}{\NLN@outfile}%
}{%
  \end{VerbatimOutAppend}%
  \immediate\write\NLN@outfile{>@@NLN@display@\the\NLN@counter}%
  \csname NLN@snippet@\the\NLN@counter\endcsname
  \global\advance\NLN@counter by 1\relax
}

\cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }

\newcommand\NLN@snippet@inlined[1]{%
  \group_begin:
  \typeout{DEBUG~PY~STYLE:<\NLN@opt@style>}
  \use_c:n { PYstyledefault }
  \tl_if_empty:NF \NLN@opt@style {
    \use_c:n { PYstyle\NLN@opt@style }
  }
  \cs_if_exist:cTF {PY} {PYOK} {PYKO}
  \NLN@opt@font
  \NLN@process@more@options{ \NLN:n { inline_method} }%
  \exp_last_unbraced:NxNo
  \use:c { \NLN:n { inline_method } } [ \NLNRemainingOptions ]{#1}%
  \group_end:
}

% ERROR: JL undefined \NLN@alllinenos

\ProvideDocumentCommand\captionof{mm}{}
\def\NLN@alllinenos{(0)}
\prg_new_conditional:Nnn \NLN_yorn:n { T, F, TF } {
  \group_begin:
  \prop_get:cnNT {g/NLN/code/} { #1 } \l_tmpa_tl {
    \exp_args:NnV
    \regex_match:nnT {^[tTyY]} \l_tmpa_tl {
      \group_end:
      \prg_return_true:
    }
  }
  \group_end:
  \prg_return_false:
}
\newenvironment{NLN@snippet@framed}{%
  \group_begin:
  \NLN@leftmargin\z@
  \NLN_yorn:nT {linenos} {
    \expandafter \NLNWidest\NLN@alllinenos{\FormatLineNumber}{\NLN@leftmargin}%
    \exp_args:NNx
    \advance\NLN@leftmargin { \NLN:n {linenosep} }
  }
  %
  \tl_clear:N \l_NLN_tl
  \NLN:nNTF {label} \l_tmpa_tl {
    \tl_set:N \l_NLN_tl {%
      \captionof{pygcode}{\label{\NLN:n {label}} \NLN:n {caption}}%
      % \nopagebreak
      \vskip -0.7\baselineskip
    }%
  } {
    \NLN:nNT {caption} \l_tmpa_tl {
      \tl_set:N \l_NLN_tl {%
        \captionof {pygcode} {\l_tmpa_tl}%
        % \nopagebreak
        \vskip -0.7\baselineskip
      }%
    }
  }
  \l_NLN_tl
  %
  \exp_args:Nx \tl_if_empty:nF { \NLN:n {boxing_method} } {
    \exp_args:Nx
    \NLN@process@more@options { \NLN:n {boxing_method} }%
    \exp_last_unbraced:NxNo
    \begin { \NLN:n {boxing_method} } [ \NLNRemainingOptions ]
  }
  \csname PYstyle\NLN@opt@style\endcsname
  \NLN@opt@font
  \noindent
} {
  \exp_args:Nx \tl_if_empty:nF { \NLN:n {boxing_method} } {
    \exp_args:Nx
    \end { \NLN:n {boxing_method} }
  }
  \group_end:
}
```

```


\def\FormatLineNumber#1{{\rmfamily\tiny#1}}


\newdimen\NLN@leftmargin
\newdimen\NLN@linenosep

\def\NLN@lineno@do#1{%
  \NLN@linenosep 0pt%
  \use:c { NLN@ \NLN:n {boxing_method} @margin }
  \exp_args:NNx
  \advance \NLN@linenosep { \NLN:n {linenosep} }
  \hbox_overlap_left:n {%
    \FormatLineNumber{#1}%
    \hspace*{\NLN@linenosep}}%
}

\newcommand\NLN@tcbox@more@options{%
  nobeforeafter,%
  tcbox~raise~base,%
  left=0mm,%
  right=0mm,%
  top=0mm,%
  bottom=0mm,%
  boxsep=2pt,%
  arc=1pt,%
  boxrule=0pt,%
  \NLN_options_if_in:nT {colback} {
    colback=\NLN:n {colback}
  }
}

\newcommand\NLN@mdframed@more@options{%
  leftmargin=\NLN@leftmargin,%
  frametitlerule=true,%
  \NLN_if_in:nT {colback} {
    backgroundcolor=\NLN:n {colback}
  }
}

\newcommand\NLN@tcolorbox@more@options{%
  grow~to~left~by=-\NLN@leftmargin,%
  \NLN_if_in:nNT {colback} {
    colback=\NLN:n {colback}
  }
}

\newcommand\NLN@boite@more@options{%
  leftmargin=\NLN@leftmargin,%
  \ifcsname NLN@opt@colback\endcsname
    colback=\NLN@opt@colback,%
  \fi
}
```

```

\newcommand\NLN@mdframed@margin{%
  \advance \NLN@linenosep \mdflength{outerlinewidth}%
  \advance \NLN@linenosep \mdflength{middlelinewidth}%
  \advance \NLN@linenosep \mdflength{innerlinewidth}%
  \advance \NLN@linenosep \mdflength{innerleftmargin}%
}

\newcommand\NLN@tcolorbox@margin{%
  \advance \NLN@linenosep \kvtcb@left@rule
  \advance \NLN@linenosep \kvtcb@leftupper
  \advance \NLN@linenosep \kvtcb@boxsep
}

\newcommand\NLN@boite@margin{%
  \advance \NLN@linenosep \boite@leftrule
  \advance \NLN@linenosep \boite@boxsep
}

\def\NLN@global@options{}

\newcommand\setpygmented[1]{%
  \def\NLN@global@options{/NLN/.cd,#1}%
}


% ==========================================================
% final actions
% ==========================================================

\AtEndOfPackage{%
  \IfFileExists{\jobname.pygmented}{%
    \input{\jobname.pygmented}%
  }{%
    \PackageWarning{inline}{File ‘\jobname.pygmented’ not found.}%
  }%
  \immediate\openout\NLN@outfile\jobname.snippets%
}

\AtEndDocument{%
  \closeout\NLN@outfile%
}
\ExplSyntaxOff
```
⟨/sty⟩