default

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = {
       'style': style,
       'page_break': page_break,
     self.\_doc = Document()
     font = self.\_doc.styles \hbox{['Normal']}. font
    font.name = font name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  def root(self) -> p.Path:
    return self._root
  def add(
     path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = \textbf{None},
  ) -> Self:
    if plain:
       {\bf return}\ self.\_add\_plain(path, page\_break, title)
     path = p.Path(path).absolute()
    code = path.read\_text()
     lexer = guess\_lexer\_for\_filename(path.name, \, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, \, \{ \, \})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get("bold", False)
       run.italic = style.get('italic', False)
```

```
run.underline = style.get(\c'underline', False)
        # color
       color = style.get(\c'color',\ None)
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self.
     path: Path,
     page\_break: t.Optional[bool] = \textbf{None}, \ title: t.Optional[bool] = \textbf{None},
  ) -> Self:
     path = p.Path(path).absolute() \\
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self.\_doc.add\_paragraph(path.read\_text())
     if page_break or self._default['page_break']:
        self.\_doc.add\_page\_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

emacs

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
     \underline{self}.\_root = p.Path(root).absolute()
     \underline{self}.\underline{-}default = \{
       'style': style,
       'page_break': page_break,
```

```
self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
   @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self.
     path: Path, plain: bool = False,
     style: t.Optional[str] = \textcolor{red}{\textbf{None}}, \ page\_break: t.Optional[bool] = \textcolor{red}{\textbf{None}},
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read\_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = \underline{self.\_}doc.add\_paragraph()
     for type, value in lexer.get_tokens(code):
        style = styles.get(type, {})
       run = paragraph.add\_run(value)
        # bold, italic, underline
       run.bold = style.get('bold', False)
        run.italic = style.get('italic', {\color{red} False})
       run.underline = style.get('underline', False)
        \# color
        color = style.get('color', None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
    page\_break: t.Optional[bool] = \textcolor{red}{\textbf{None}}, \ title: t.Optional[bool] = \textcolor{red}{\textbf{None}},
  ) -> Self:
     path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
        \underline{self}.\_doc.add\_page\_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root=\text{'.'}, page\_break=\text{False}, font\_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
```

friendly

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = \__qualname\__
  def __init__(
    self.
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
    self.\_doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font\_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
     self,
    path: Path, plain: bool = False,
    style: t.Optional[str] = \textbf{None}, page\_break: t.Optional[bool] = \textbf{None},
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
```

```
# bold, italic, underline
        run.bold = style.get('bold', \textbf{False})
        run.italic = style.get('italic', False)
       run.underline = style.get('underline', {\color{red} False})
        # color
       color = style.get(\c'color',\c None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  \label{eq:continuous} \textbf{def} \ save(self, \ path: \ Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self._doc.add_paragraph(path.read_text())
     if \ page\_break \ or \ self.\_default['page\_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break= \textbf{False}, font\_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

colorful

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font\_name: str = "Times New Roman", font\_size: int = \textbf{9},
    self.\_root = p.Path(root).absolute()
```

```
self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font\_name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
   @property
  def root(self) -> p.Path:
     return self._root
  def add(
    self,
    path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', {\color{red} False})
       # color
       color = style.get('color',\, None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if \ page\_break \ or \ self.\_default['page\_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
    self.
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
    path = p.Path(path).absolute()
     self.\_doc.add\_heading(title~or~path.relative\_to(self.\_root).as\_posix(),~\textbf{0})
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
```

```
for style in get_all_styles():
   word.add(__file__, style=style, title=style)
word.save('demo.docx')
```

autumn

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import \underline{pathlib} as \underline{p}
import typing as t
from docx import Document
from\ \underline{docx.shared}\ import\ Pt,\ RGBColor
from pygments.lexers import guess_lexer_for_filename
from <a href="mailto:pygments.styles">pygments.styles</a> import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self.
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
     self._root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = \underline{self.\_doc.styles['Normal']}.font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page\_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
```

```
style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get("bold", False)
       run.italic = style.get('italic', False)
       run.underline = style.get(\c'underline', False)
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     if page_break or self._default['page_break']:
       \underline{self.}\_doc.add\_page\_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ 0)
     self._doc.add_paragraph(path.read_text())
     if\ page\_break\ or\ self.\_default \cite{break'}]:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break=False, font\_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

murphy

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
""
import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
    Self = __qualname__

def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
```

```
) -> None:
     self.\_root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
     self.\_doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @\,property
  def root(self) -> p.Path:
    return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', {\color{red} False})
       run.underline = style.get('underline', {\color{red} False})
       # color
       color = style.get('color', None)
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
      \textbf{if} \ page\_break \ \textbf{or} \ self.\_default['page\_break']: \\
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute() \\
     self.\_doc.add\_heading(title \ or \ path.relative\_to(self.\_root).as\_posix(), \ \textbf{0})
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
```

```
word = Word.new(root='.', page_break=False, font_size=7)
for style in get_all_styles():
  word.add(__file__, style=style, title=style)
word.save('demo.docx')
```

manni

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
    self.\_doc = Document()
     font = self.\_doc.styles \hbox{['Normal']}. font
     font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = \textbf{None},
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
    code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
```

```
paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get(\bold', \begin{center} \textbf{False} \end{center})
       run.italic = style.get('italic', {\color{red} False})
       run.underline = style.get(\c'underline', \c False)
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(\_\_file\_\_, style=style, title=style)
  word.save('demo.docx')
```

material

```
""

pygments = "^2.12.0"

python-docx = "^0.8.11"

""

import pathlib as p

import typing as t

from docx import Document

from docx.shared import P, RGBColor

from pygments.lexers import mass lexer for filenant

from pygments.styles import acceptate by mans.

Path = 1.0 more [str, p.Path]

class Word:

Self = __qualname__

def __init__(
```

```
root: Path = '.', style: str = 'default', page_break: bool = True,
     ont_name: str = 'Times New Roman', font_size: int = 9,
@classmethod
def new(cls, *args, **kwargs) -> Seld:
return cls(*args, **kwargs)
@property
def root(self) -> p.Path:
  return self._ro
def add(
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
) -> Self:
if plain:
    return self._add_plain(path, page_break, title)
   code = path.read tex.()
lexer = guess.lexer.for_filename(path.mane, code)
styles = dict(cot.style_by_mane(style_or_self._defaul.['style']))
   # heading
   self. doc.aid heading(tite or path.relative_tr(self. roor).as_pose(), 0) # paragraph
   # paragraph
   paragraph = self. doc.add_paragraph() for type, value in lexer.get_tokens(code):
     style = styles.get(type, {})
run = paragraph.add_run(value)
      # bold, italic, underline
      # color
      if color is not None:
   # page break
   if page_break or self._default['page_break']:
   return self
def save(self, path: Path) -> Self:
   return self
def _add_plain(
   path: Path,
page, bread: (.Optional[bool] = None, title: (.Optional[bool] = None,
   self. do. add headin (into or path. ciative in (self. root), as post (), 0) self. do. add pumpraph (sath. cod text()) if sage break or self. defaul ['page_break']: self. defauld page_break() return self
   return self
```

```
if __name__ == '__main__':
    from pygments.styles import geneall styles

word = Word.new(con=',', page breat =False, four size =7)
for style in geneall styles():
    word.new('file__, style=style, title=style)
    word.new('demo.docx')
```

monokai

```
import pathlib as p
import typing as
from docx import Document
from lock shared import Pt. RGBColor
from sygments lokers import guess loker for filename
from sygments styles import get style by mane
Path = t.Union[str, p.Path]
class Word:
   def __init__(
      root: Path = '.', style: str = 'default', page_break: bool = True font_name: str = 'Times New Roman', font_size: int = 9,
    -> None:
     self\_root = p.Path(root).absolute()
      self._default = {
   'style': style,
      font = self._doc.styles['Normal'].font
      font.name = font_name
      font.size = Pt(font_size)
   @classmethod
   def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
   @property
   def root(self) -> p.Path:
     return self._root
   def add(
      path: Path, plain: bool = False,
     style: (.Optional[str] = None, page_break: (.Optional[bool] = None title: (.Optional[str] = None,
     if plain:
      return self_add_plain(path, page_break, title)
path = p.Path(path).absolute()
      code = path.read_text()
      lexer = guess_lexer_for_filename(path.name, code)
       styles = dict(get_style_by_name(style or self._default['style']))
      # heading
```

```
elf._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()

for type, value in lexer.get_tokens(code)
       style = styles.get(type, { })
        # bold, italic, underline
        run.bold = style.get('bold', False)
        run.italic = style.get('italic', False)
        run.underline = style.get('underline', False)
         color = style.get('color', None)
        if color is not None:
         run.font.color.rgb = RGBColor.from_string(color)
     # page break
     if page_break or self._default['page_break']
  def save(self, path: Path) -> Self:
  def _add_plain(
   page_break: t.Optional[bool] = None, title: t.Optional[bool] = None
t-> Self:
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ 0)
     self_doc.add_paragraph(path.read_text())
if page_break or self_default|'page_break'|
if __name__ == '__main__'
  from pygments.styles import get_all_styles
   word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
```

perldoc

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
Self = __qualname__
```

```
def __init__(
  self.
  root: Path = '.', style: str = 'default', page_break: bool = True,
  font_name: str = 'Times New Roman', font_size: int = 9,
) -> None:
  \textcolor{red}{\textbf{self.\_root}} = p.Path(root).absolute()
  self.\_default = \{
     'style': style,
     'page_break': page_break,
  self._doc = Document()
  font = \underline{self.\_doc.styles['Normal']}.font
  font.name = font_name
  font.size = Pt(font_size)
@classmethod
def new(cls, *args, **kwargs) -> Self:
  return cls(*args, **kwargs)
@property
def root(self) -> p.Path:
  return self._root
def add(
  self,
  path: Path, plain: bool = False,
  style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
  title: t.Optional[str] = None,
) -> Self:
  if plain:
     return self._add_plain(path, page_break, title)
  path = p.Path(path).absolute()
  code = path.read_text()
  lexer = guess_lexer_for_filename(path.name, code)
  styles = dict(get_style_by_name(style or self._default['style']))
  # heading
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  # paragraph
  paragraph = self._doc.add_paragraph()
  for type, value in lexer.get_tokens(code):
     style = styles.get(type, {})
     run = paragraph.add\_run(value)
     # bold, italic, underline
     run.bold = style.get('bold', False)
     run.italic = style.get('italic', False)
     run.underline = style.get('underline', {\bf False})
     # color
     color = style.get(\c'color',\c None)
     if color is not None:
       run.font.color.rgb = RGBColor.from\_string(color)
   # page break
  if page_break or self._default['page_break']:
     self._doc.add_page_break()
  return self
def save(self, path: Path) -> Self:
  self._doc.save(path)
  return self
def _add_plain(
  self.
  path: Path,
  page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
) -> Self:
  path = p.Path(path).absolute()
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  self._doc.add_paragraph(path.read_text())
  if page_break or self._default['page_break']:
     self._doc.add_page_break()
   return self
```

```
if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
    for style in get_all_styles():
        word.add(_file__, style=style, title=style)
    word.save('demo.docx')
```

pastie

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = "Times New Roman", font_size: int = 9,
  ) -> None:
    self.\_root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
    self._doc = Document()
     font = self.\_doc.styles \cite{Normal'}.font
     font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    path: Path, plain: bool = \textbf{False},
    style: t.Optional[str] = \textbf{None}, page\_break: t.Optional[bool] = \textbf{None},
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
```

```
styles = dict(get_style_by_name(style or self._default['style']))
             # heading
             self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(), \textcolor{red}{0})
             # paragraph
             paragraph = self._doc.add_paragraph()
             for type, value in lexer.get_tokens(code):
                   style = styles.get(type, {})
                  run = paragraph.add\_run(value)
                   # bold, italic, underline
                  run.bold = style.get(\bold', False)
                   run.italic = style.get('italic', False)
                  run.underline = style.get(\c'underline', False)
                   # color
                  color = style.get(\begin{subarray}{c} color = style.get(\begin{subar
                   if color is not None:
                        run.font.color.rgb = RGBColor.from\_string(color)
             if page_break or self._default['page_break']:
                  self._doc.add_page_break()
             return self
      def save(self, path: Path) -> Self:
             self._doc.save(path)
             return self
      def\_add\_plain(
            self,
            path: Path,
           page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
      ) -> Self:
            path = p.Path(path).absolute()
            self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(), \\ \textbf{0})
             self._doc.add_paragraph(path.read_text())
            if\ page\_break\ or\ self.\_default \cite{break'}]:
                  self._doc.add_page_break()
             return self
if __name__ == '__main___':
      from pygments.styles import get_all_styles
      word = Word.new(root='.', page\_break=False, font\_size=7)
      for style in get_all_styles():
           word.add(__file__, style=style, title=style)
      word.save('demo.docx')
```

borland

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
```

```
Self = \__qualname\__
def __init__(
  self.
  root: Path = '.', style: str = 'default', page_break: bool = True,
  font_name: str = 'Times New Roman', font_size: int = 9,
) -> None:
  self._root = p.Path(root).absolute()
  self._default = {
     'style': style,
     'page_break': page_break,
  self._doc = Document()
  font = self.\_doc.styles \cite{Normal'}.font
  font.name = font_name
  font.size = Pt(font_size)
@classmethod
def new(cls, *args, **kwargs) -> Self:
  return cls(*args, **kwargs)
@property
def root(self) -> p.Path:
  return self._root
def add(
  self,
  path: Path, plain: bool = False,
  style: t.Optional[str] = \textbf{None}, page\_break: t.Optional[bool] = \textbf{None},
  title: t.Optional[str] = None,
) -> Self:
  if plain:
     return self._add_plain(path, page_break, title)
  path = p.Path(path).absolute()
  code = path.read_text()
  lexer = guess\_lexer\_for\_filename(path.name, code)
  styles = dict(get_style_by_name(style or self._default['style']))
  # heading
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  # paragraph
  paragraph = self._doc.add_paragraph()
  for type, value in lexer.get_tokens(code):
     style = styles.get(type, {})
     run = paragraph.add\_run(value)
     # bold, italic, underline
     run.bold = style.get(\mbox{'bold'}, \mbox{\bf False})
     run.italic = style.get('italic', False)
     run.underline = style.get('underline', {\color{red} False})
     # color
     color = style.get('color', None)
     if color is not None:
       run.font.color.rgb = RGBColor.from_string(color)
   # page break
  if \ page\_break \ or \ self.\_default['page\_break']:
     self._doc.add_page_break()
  return self
def save(self, path: Path) -> Self:
  self._doc.save(path)
  return self
def _add_plain(
  self.
  path: Path,
  page\_break: t.Optional[bool] = \textbf{None}, \ title: t.Optional[bool] = \textbf{None},
) -> Self:
  path = p.Path(path).absolute()
   self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  self._doc.add_paragraph(path.read_text())
  if page_break or self._default['page_break']:
```

```
self._doc.add_page_break()
  return self

if __name__ == '__main__':
  from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

trac

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from \ {\tt pygments.styles} \ import \ {\tt get\_style\_by\_name}
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self.
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
     self \_root = p \_Path(root) \_absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
     self_{-}doc = Document()
     font = self_doc_styles['Normal']_font
     font.name = font_name
     font.size = Pt(font_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     \textbf{return} \ self.\_root
  def add(
     self,
     path: Path, plain: bool = False,
     style: \textbf{t.}Optional[str] = \textbf{None}, page\_break: \textbf{t.}Optional[bool] = \textbf{None},
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
```

```
code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph \centerdot add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', \textbf{False})
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       color = style \cdot get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     {\tt self\_doc.add\_heading(title~or~path\_relative\_to(self\_root).as\_posix(),~0)}
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from \ {\tt pygments.styles} \ import \ {\tt get\_all\_styles}
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(\underline{file}, style=style, title=style)
  word.save('demo.docx')
```

native

```
""

pygments = "^2.12.0"

python-docx = "^0.8.11"

""

import pathlib as p
import typing as t

from docx import Document

from docx.shared import Pt, RGBColor

from pygments.lexers import guess_lexer_for_filename

from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]
```

```
class Word:
  def __init__(
     self.
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     \boldsymbol{self}.\_root = p.Path(root).absolute()
     self.\_default = \{
        'style': style,
        'page_break': page_break,
     self._doc = Document()
     font = \boldsymbol{self}.\_doc.styles[\textbf{'Normal'}].font
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
     if plain:
       return self._add_plain(path, page_break, title)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
        style = styles.get(type, { })
        # bold, italic, underline
        run.bold = style.get(\textbf{'bold'},\,\textbf{False})
        run.italic = style.get('italic', False)
        run.underline = style.get(\hbox{\ensuremath{^{'}}{}} underline\hbox{\ensuremath{^{''}}{}}, \textbf{\textit{False}})
        # color
        color = style.get(\c'color',\ None)
        if color is not None:
     # page break
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
  \pmb{\text{def save}(\text{self},\,\text{path}:\,\text{Path})} \mathrel{->} \text{Self:}
     self._doc.save(path)
     return self
  def _add_plain(
     self.
     page\_break: t.Optional[\textbf{bool}] = \textbf{None}, title: t.Optional[\textbf{bool}] = \textbf{None},
     self.\_doc.add\_heading(title \ or \ path.relative\_to(self.\_root).as\_posix(), \ 0)
```

```
self._doc.add_paragraph(path.read_text())
if page_break or self._default('page_break'):
    self._doc.add_page_break()
    return self

if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
for style in get_all_styles():
    word.add(_file__, style=style, title=style)
word.save('demo.docx')
```

fruity

if

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import
             as
import
from
from
               import
from
                   import
from
                   import
class
       __qualname__
  def __init__
                           'default'
                                                   True
                                                   9
                  'Times New Roman'
     None
      'style'
      'page_break'
                        'Normal'
  def new
    return
  def root
    return
  def add
                         False
                        None
                                                         None
                       None
```

```
return
                                                       'style'
     # heading
                                                                     0
    # paragraph
     for
       \# \ bold, \ italic, \ underline
                         'bold' False
                          'italic' False
                              'underline' False
       # color
                       'color' None
       if
                    None
    # page break
    if
                                 'page_break'
     return
  def save
    return
  def\_add\_plain
                                  None
                                                               None
                                                                     0
    if
                                 'page_break'
    return
if __name__
               '__main__'
  from
                       import
                                       False
                                                        7
  for
               __file__
            'demo.docx'
```

bw

```
""

pygments = "^2.12.0"

python-docx = "^0.8.11"

""

import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
```

```
class Word:
  Self = __qualname__
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self._doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     path:\ Path,\ plain:\ bool=\textbf{False},
     style: t.Optional[str] = \textbf{None}, page\_break: t.Optional[bool] = \textbf{None},
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get(\textit{'underline'}, \, False)
       # color
       color = style.get(\textit{'color'}, \textbf{None})
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     page\_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
```

```
path = p.Path(path).absolute()
self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
self._doc.add_paragraph(path.read_text())
if page_break or self._default['page_break']:
    self._doc.add_page_break()
    return self

if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
for style in get_all_styles():
    word.add(__file__, style=style, title=style)
word.save('demo.docx')
```

vim

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
    {\bf self.\_root = p.Path(root).absolute()}
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  def root(self) -> p.Path:
    return self._root
  def add(
    self.
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
```

```
) -> Self:
    if plain:
      return self._add_plain(path, page_break, title)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       # bold, italic, underline
       {\rm run.bold} = {\rm style.get}(\textbf{'bold'},\, \textbf{False})
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
    self._doc.save(path)
     return self
  def _add_plain(
    self,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

VS

```
""
pygments = "^2.12.0"
python-docx = "^0.8.11"
""
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
```

```
Path = t.Union[str, p.Path]
```

```
class Word:
  Self = \underline{\hspace{0.3cm}} qualname\underline{\hspace{0.3cm}}
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get("bold", False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)\\
       # color
       color = style.get('color', None)
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self.\_doc.add\_page\_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
```

```
page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
) -> Self:
    path = p.Path(path).absolute()
    self__doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    self__doc.add_paragraph(path.read_text())
    if page_break or self__default['page_break']:
        self__doc.add_page_break()
    return self

if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
for style in get_all_styles():
    word.add(__file__, style=style, title=style)
    word.save('demo.docx')
```

tango

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self.
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
    self \_root = p \_Path(root) \_absolute()
    self._default = {
       'style': style,
       'page_break': page_break,
    self._doc = Document()
    font = self.\_doc.styles['Normal'].font
    font.name = font_name
    font_size = Pt(font_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
    path: Path, plain: bool = False,
```

```
style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
    self,
    path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self._doc.add_paragraph(path.read_text())
    if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

rrt

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
"import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
```

```
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
     path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
    self._doc.save(path)
     return self
  def _add_plain(
```

```
self,
    path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
    path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self.\_doc.add\_paragraph(path.read\_text())
    if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

xcode

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self.
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
    self.\_root = p.Path(root).absolute()
    self._default = {
       'style': style,
       'page_break': page_break,
    self._doc = Document()
    font = self.\_doc.styles['Normal'].font
    font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
```

```
self,
           path: Path, plain: bool = False,
           style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
          title: t.Optional[str] = None,
     ) -> Self:
          if plain:
                 return self._add_plain(path, page_break, title)
           path = p.Path(path).absolute()
           code = path.read_text()
           lexer = guess\_lexer\_for\_filename(path.name, code)
           styles = dict(get_style_by_name(style or self._default['style']))
           self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
           # paragraph
           paragraph = self._doc.add_paragraph()
           for type, value in lexer.get_tokens(code):
                 style = styles.get(type, { })
                run = paragraph.add\_run(value)
                 # bold, italic, underline
                 run.bold = style.get(\bold', False)
                 run.italic = style.get('italic', False)
                 run.underline = style.get('underline', False)
                color = style.get('color', None)
                 if color is not None:
                     run.font.color.rgb = RGBColor.from\_string(color)
            # page break
           if page_break or self._default['page_break']:
                self._doc.add_page_break()
           return self
     def save(self, path: Path) -> Self:
           self._doc.save(path)
          return self
     def _add_plain(
          self,
          path: Path,
          page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
     ) -> Self:
          path = p.Path(path).absolute()
           self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(), \\ 0)
           self._doc.add_paragraph(path.read_text())
          if page_break or self._default['page_break']:
                self._doc.add_page_break()
           return self
if __name__ == '__main__':
     from pygments.styles import get_all_styles
     word = Word.new(root=\colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon="colon=
     for style in get_all_styles():
           word.add(\__file\_\_, style=style, title=style)
     word.save('demo.docx')
```

igor

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
"import pathlib as p
import typing as t
```

```
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
    code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
```

```
def_add_plain(
    self,
    path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
) -> Self:
    path = p.Path(path).absolute()
    self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    self._doc.add_paragraph(path.read_text())
    if page_break or self._default['page_break']:
        self._doc.add_page_break()
    return self

if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
    for style in get_all_styles():
        word.add(_file__, style=style, title=style)
        word.save('demo.docx')
```

paraiso-light

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
    self.\_default = \{
       'style': style,
       'page_break': page_break,
    self._doc = Document()
    font = self.\_doc.styles \hbox{['Normal']}.font
     font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  def root(self) -> p.Path:
    return self._root
```

```
def add(
     self,
    path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
    self._doc.save(path)
     return self
  def _add_plain(
     self,
    path: Path.
    page\_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
    path = p.Path(path).absolute()
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(), \\ \textcolor{red}{0})
     self._doc.add_paragraph(path.read_text())
    if\ page\_break\ or\ self.\_default['page\_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break=False, font\_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

paraiso-dark

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
"import pathlib as p
```

```
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
class Word:
 Self = __qualname__
  def __init__(
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
      'style': style,
       'page_break': page_break,
    font = self._doc.styles['Normal'].font
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
    if plain
      return self._add_plain(path, page_break, title)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     for type, value in lexer.get_tokens(code):
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
     # page break
     if page_break or self._default['page_break']:
     return self
```

def save(self, path: Path) -> Self:

```
self._doc.save(path)
return self

def _add_plain(
    self,
    path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
) -> Self:
    path = p.Path(path).absolute()
    self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    self._doc.add_paragraph(path.read_text())
    if page_break or self._default['page_break']:
        self._doc.add_page_break()
    return self

if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
for style in get_all_styles():
    word.add(_file__, style=style, title=style)
word.save('demo.docx')
```

lovelace

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     \textit{self}.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = \mathit{self}.\_doc.styles \cite{block} 'Normal']. font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return \ \mathit{cls}(*args, \ **kwargs)
  @property
```

```
def \ root(self) \rightarrow p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, \ page\_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
     if plain:
        return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
     code = path.read\_text()
     lexer = guess\_lexer\_for\_filename(path.name,\ code)
     styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
        style = styles.get(type, \{ \})
        run = paragraph.add\_run(value)
        # bold, italic, underline
        run.bold = style.get('bold', False)
        run.italic = style.get('italic', \mathit{False})
        run.underline = style.get( \textit{`underline'}, \textit{False})
        # color
        color = style.get('color', None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     if\ page\_break\ or\ \textit{self}.\_default['page\_break']:
        self.\_doc.add\_page\_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute() \\
     \textit{self}.\_doc.add\_heading(title \ or \ path.relative\_to(\textit{self}.\_root).as\_posix(), \ 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break=\mathit{False}, font\_size=7)
  for style in get_all_styles():
     word.add(\underline{\phantom{a}file}\underline{\phantom{a}},\, style = style,\, title = style)
  word.save('demo.docx')
```

algol

```
""

pygments = "^2.12.0"

python-docx = "^0.8.11"
```

```
import pathlib as p
import typing as t
from docx import Document
\underline{\textbf{from}}\ docx.shared\ \underline{\textbf{import}}\ \textbf{Pt}, \textbf{RGBColor}
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> <u>None</u>:
     self._root = p.Path(root).absolute()
     self._default = {
        'style': style,
        'page_break': page_break,
     self._doc = Document()
     font = self._doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  <u>def</u> root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = \underline{False},
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = \textit{dict}(\texttt{get\_style\_by\_name}(\texttt{style or } \textit{self}.\_\texttt{default['style']}))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \#\,paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
        style = styles.get(type, { })
        run = paragraph.add\_run(value)
        # bold, italic, underline
        run.bold = style.get('bold', \underline{False})
       run.italic = style.get('italic', \underline{False})
        run.underline = style.get('underline', \underline{False})
        # color
        color = style.get('color', None')
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       \textit{self}.\_doc.add\_page\_break()
     return self
```

```
def save(self, path: Path) -> Self:
      self._doc.save(path)
      \underline{\mathbf{return}}\ \mathbf{\mathit{self}}
   \underline{\mathbf{def}}\,\_add\_plain(
      self,
      path: Path,
      page\_break: t.Optional[\textit{bool}] = \underline{None}, title: t.Optional[\textit{bool}] = \underline{None},
   ) -> Self:
      path = p.Path(path).absolute()
      \textit{self}._doc.add_heading(title or path.relative_to(\textit{self}._root).as_posix(), 0)
      self._doc.add_paragraph(path.read_text())
      \underline{\textbf{if}} \ page\_break \ \textbf{or} \ \textbf{\textit{self}}.\_default[\textit{'page\_break'}]:
          self._doc.add_page_break()
       return self
<u>if</u> __name__ == '__main__':
   \underline{from}\ \textit{pygments.styles}\ \underline{import}\ get\_all\_styles
   word = Word.new(root='.', page_break=<u>False</u>, font_size=7)
   for style in get_all_styles():
      word.add(\_\mathit{file}\_\_, style=style, title=style)
   word.save('demo.docx')
```

algol_nu

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
\textbf{from } \textit{docx.shared } \textbf{import} \ \mathsf{Pt}, \ \mathsf{RGBColor}
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
     \textit{self}.\_root = p.Path(root).absolute()
     self._default = {
        'style': style,
        'page_break': page_break,
     self._doc = Document()
     font = \textit{self}.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
```

```
@property
  \textbf{def } root(\textbf{\textit{self}}) \rightarrow \text{p.Path:}
     return self._root
  def add(
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
     if plain:
        return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
        style = styles.get(type, { })
        run = paragraph.add\_run(value)
        # bold, italic, underline
        run.bold = style.get('bold', \textbf{False})
        run.italic = style.get('italic', False)
        run.underline = style.get(\textit{'underline'}, \, False)
        # color
        color = style.get('color', None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     \mathbf{return}\ self
  \mathbf{def}\,\_add\_plain(
     self,
     path: Path,
     page\_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     \textit{self}.\_\texttt{doc.add\_heading}(title \ \textbf{or} \ path.relative\_to(\textit{self}.\_root).as\_posix(), \ 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
        \textit{self}.\_\texttt{doc.add}\_\texttt{page}\_\texttt{break}()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(\_\mathit{file}\_\_, style=style, title=style)
  word.save('demo.docx')
```

arduino

```
pygments = "^2.12.0"
python\text{-}docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self._root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self._doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
    path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from_string(color)
     # page break
```

```
if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
    self.
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
    path = p.Path(path).absolute()
    self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    self._doc.add_paragraph(path.read_text())
    if page_break or self._default['page_break']:
       self._doc.add_page_break()
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

rainbow_dash

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
    self._root = p.Path(root).absolute()
    self.\_default = \{
       'style': style,
       'page_break': page_break,
    self._doc = Document()
    font = self._doc.styles['Normal'].font
    font.name = font\_name
    font.size = Pt(font_size)
```

```
@classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
    self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
    path = p.Path(path).absolute()
     code = path.read\_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = \textbf{dict}(get\_style\_by\_name(style \ \textbf{or} \ \textbf{self}.\_default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
    self,
    path: Path,
    page\_break: t.Optional[\textbf{bool}] = \textbf{None}, \ title: t.Optional[\textbf{bool}] = \textbf{None},
  ) -> Self:
     path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \textcolor{red}{\textbf{self}}.\_doc.add\_paragraph(path.read\_text())
     if page_break or self._default['page_break']:
       \textcolor{red}{\textbf{self}}.\_\texttt{doc}.\texttt{add}\_\texttt{page}\_\texttt{break}()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

abap

```
pygments = "^2.12.0"
python\text{-}docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
    font.name = font name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
   @property
  def root(self) -> p.Path:
    return self._root
  def add(
     path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
    code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     self.\_doc.add\_heading(title \ or \ path.relative\_to(self.\_root).as\_posix(), \ 0)
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, \{\})
       run = paragraph.add_run(value)
       \#\ bold,\ italic,\ underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
```

```
run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self.
     path: Path,
     page\_break: t.Optional[bool] = \\ \underbrace{None}, title: t.Optional[bool] = \\ \underbrace{None},
  ) -> Self:
     path = p.Path(path).absolute() \\
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self.\_doc.add\_paragraph(path.read\_text())
     if page_break or self._default['page_break']:
       self.\_doc.add\_page\_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

solarized-dark

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from\ pygments.lexers\ import\ guess\_lexer\_for\_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname_
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
    self._root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
```

```
self._doc = Document()
    font = self.\_doc.styles['Normal'].font
    font.name = font name
    font.size = Pt(font_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self.
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page\_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
    path = p.Path(path).absolute()
    code = path.read\_text()
    lexer = guess_lexer_for_filename(path.name, code)
    styles = dict(get_style_by_name(style or self._default['style']))
    self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    # paragraph
    paragraph = self.\_doc.add\_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
    if page_break or self._default['page_break']:
       self._doc.add_page_break()
    return self
  def save(self, path: Path) -> Self:
    self._doc.save(path)
    return self
  def _add_plain(
    self.
    path: Path,
    page\_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
    path = p.Path(path).absolute()
    self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
    self._doc.add_paragraph(path.read_text())
    if page_break or self._default['page_break']:
       self._doc.add_page_break()
    return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root=\text{'.'}, page\_break=\text{False}, font\_size=\text{7})
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
```

solarized-light

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
    self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
    self.\_doc = Document()
    font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
      return self._add_plain(path, page_break, title)
    path = p.Path(path).absolute() \\
    code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
    styles = dict(get_style_by_name(style or self._default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
```

```
# bold, italic, underline
        run.bold = style.get('bold', False)
        run.italic = style.get('italic', False)\\
        run.underline = style.get('underline', False)
        \# color
        color = style.get('color',\ None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
      # page break
     if page_break or self._default['page_break']:
        self._doc.add_page_break()
     return self
  \label{eq:continuous} \mbox{def save}(\mbox{self, path: Path}) -> \mbox{Self:}
     self._doc.save(path)
     return self
  \underline{def}\,\underline{-}add\underline{-}plain(
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     self.\_doc.add\_heading(title \  \, \underbrace{or} \  \, path.relative\_to(self.\_root).as\_posix(), \  \, 0)
     self._doc.add_paragraph(path.read_text())
     if\ page\_break\ or\ self.\_default['page\_break']:
        self._doc.add_page_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
  word = Word.new(root=\text{'.'}, page\_break=False, font\_size=7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

sas

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font\_name: str = "Times New Roman", font\_size: int = 9,
    self.\_root = p.Path(root).absolute()
```

```
self.\_default = \{
       'style': style.
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font\_name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
     title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess_lexer_for_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \# paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', {\color{red} False})
       # color
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  \underline{def}\,\underline{-}add\underline{-}plain(
     self.
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
     path = p.Path(path).absolute()
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ \textbf{0})
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
```

```
for style in get_all_styles():
  word.add(__file__, style=style, title=style)
word.save('demo.docx')
```

stata

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     self,
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
     return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
     self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = \textcolor{red}{\textbf{None}}, \ page\_break: t.Optional[bool] = \textcolor{red}{\textbf{None}},
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, \, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ \textbf{0})
     \#\,paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
```

```
style = styles.get(type, {})
        run = paragraph.add_run(value)
        # bold, italic, underline
       run.bold = style.get(\buildrel{'bold'}, \begin{center} \textbf{False} \end{center})
        run.italic = style.get('italic', False)
        run.underline = style.get('underline', {\color{red} False})
        # color
       color = style.get('color', None)
        if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     if page_break or self._default['page_break']:
       self.\_doc.add\_page\_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
     path = p.Path(path).absolute()
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ 0)
     self._doc.add_paragraph(path.read_text())
     if \ page\_break \ or \ self.\_default['page\_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break= \pmb{False}, font\_size= 7)
  for style in get_all_styles():
     word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

stata-light

```
pygments = "^2.12.0"
python-docx = "^0.8.11"

import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
    Self = __qualname__

def __init__(
    self,
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
```

```
) -> None:
     self.\_root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
     self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
     font.size = Pt(font_size)
   @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    self,
     path: Path, plain: bool = False,
     style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
    code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', {\color{red} False})
       # color
       color = style.get('color', None)
       if color is not None:
          run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
    path = p.Path(path).absolute()
     self.\_doc.add\_heading(title\ or\ path.relative\_to(self.\_root).as\_posix(),\ \textbf{0})
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main___':
  from pygments.styles import get_all_styles
```

```
word = Word.new(root=', page_break=False, font_size=7)
for style in get_all_styles():
  word.add(_file__, style=style, title=style)
word.save('demo.docx')
```

stata-dark

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
     root: Path = '.', style: str = 'default', page_break: bool = True,
     font_name: str = 'Times New Roman', font_size: int = 9,
     self.\_root = p.Path(root).absolute()
     self.\_default = \{
       'style': style,
       'page_break': page_break,
    self._doc = Document()
     font = self.\_doc.styles['Normal'].font
     font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
     return self._root
  def add(
    self,
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
     if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute()
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
     styles = dict(get\_style\_by\_name(style \ or \ self.\_default['style']))
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     \#\,paragraph
```

```
paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, { })
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get('bold', False)
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', {\color{red} False})
       color = style.get('color', None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     # page break
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
    path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
    path = p.Path(path).absolute()
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page_break=False, font_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```

inkpot

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
""
import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
    Self = __qualname__
    def __init__(
```

```
root: Path = '.', style: str = 'default', page_break: bool = True,
  font_name: str = 'Times New Roman', font_size: int = 9,
) -> None:
  self._root = p.Path(root).absolute()
  self._default = {
     'page_break': page_break,
  self._doc = Document()
  font = {\color{red} self.\_doc.styles['Normal'].font}
  font.name = font_name
  font.size = Pt(font_size)
@classmethod
def new(cls, *args, **kwargs) -> Self:
  return cls(*args, **kwargs)
@property
def root(self) -> p.Path:
  return self._root
def add(
  path: Path, plain: bool = False,
  style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
  title: t.Optional[str] = None,
) -> Self:
  if plain:
    return self._add_plain(path, page_break, title)
  path = p.Path(path).absolute()
  code = path.read_text()
  lexer = guess_lexer_for_filename(path.name, code)
  styles = dict(get_style_by_name(style or self._default['style']))
  # heading
  \textcolor{red}{\textbf{self.\_}} \textbf{doc.add\_heading(title or path.relative\_to(\textcolor{red}{\textbf{self.\_}} \textbf{root).as\_posix(), 0)}
  # paragraph
  paragraph = self._doc.add_paragraph()
  for type, value in lexer.get_tokens(code):
     style = styles.get(type, { })
     run = paragraph.add_run(value)
     # bold, italic, underline
     run.bold = style.get('bold', False)
     run.italic = style.get('italic', False)
     run.underline = style.get('underline', False)
     # color
     color = style.get('color', None)
     if color is not None:
       run.font.color.rgb = RGBColor.from\_string(color)
  # page break
  if page_break or self._default['page_break']:
    self._doc.add_page_break()
  return self
def save(self, path: Path) -> Self:
  return self
def _add_plain(
  path: Path,
  page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
) -> Self:
  path = p.Path(path).absolute()
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  if page_break or self._default['page_break']:
    self._doc.add_page_break()
  return self
```

```
if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root='.', page_break=False, font_size=7)
    for style in get_all_styles():
        word.add(_file__, style=style, title=style)
    word.save('demo.docx')
```

zenburn

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
     root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
  ) -> None:
      'style': style,
       'page_break': page_break,
     font = self._doc.styles['Normal'].font
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
     styles = \frac{dict}{(get\_style\_by\_name(style\ or\ self.\_default[\text{'style'}]))}
     # heading
```

```
# paragraph
        # bold, italic, underline
        run.bold = style.get(\textbf{'bold'}, \textbf{False})
        run.italic = style.get('italic', False)
        run.underline = style.get(\hbox{\bf 'underline'}, False)
        color = style.get(\mbox{'color'}, \mbox{None})
        if color is not None:
      if page_break or self._default['page_break']:
     page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
     if page_break or self._default['page_break'].
if __name__ == '__main__':
  word = Word.new(root=\text{'.'}, page\_break=\text{False}, font\_size=\text{7})
   word.save('demo.docx')
```

gruvbox-dark

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
"import pathlib as p
import typing as t

from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name

Path = t.Union[str, p.Path]

class Word:
Self = __qualname__
```

```
def __init__(
  self.
  root: Path = '.', style: str = 'default', page_break: bool = True,
  font_name: str = 'Times New Roman', font_size: int = 9,
) -> None:
  \underline{self}.\_root = p.Path(root).absolute()
  self.\_default = \{
    'style': style.
     'page_break': page_break,
  }
  self._doc = Document()
  font = \underline{self.}\_doc.styles \hbox{['Normal']}.font
  font.name = font_name
  font.size = Pt(font_size)
@classmethod
def new(cls, *args, **kwargs) -> Self:
  return cls(*args, **kwargs)
@property
def root(self) -> p.Path:
  return self._root
def add(
  self,
  path: Path, plain: bool = False,
  style: t.Optional[str] = None, page_break: t.Optional[bool] = None,
  title: t.Optional[str] = None,
) -> Self:
  if plain:
    return self._add_plain(path, page_break, title)
  path = p.Path(path).absolute()
  code = path.read_text()
  lexer = guess_lexer_for_filename(path.name, code)
  styles = dict(get_style_by_name(style or self._default['style']))
  # heading
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  \#\,paragraph
  paragraph = self._doc.add_paragraph()
  for type, value in lexer.get_tokens(code):
    style = styles.get(type, {})
    run = paragraph.add\_run(value)
     # bold, italic, underline
    run.bold = style.get('bold', False)
    run.italic = style.get('italic', False)
     run.underline = style.get('underline', False)
     # color
    color = style.get('color', None)
     if color is not None:
       run.font.color.rgb = RGBColor.from\_string(color)
   # page break
  if page_break or self._default['page_break']:
    self._doc.add_page_break()
  return self
def save(self, path: Path) -> Self:
  self._doc.save(path)
  return self
def _add_plain(
  self.
  path: Path,
  page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  path = p.Path(path).absolute()
  self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
  self._doc.add_paragraph(path.read_text())
  if page_break or self._default['page_break']:
    self._doc.add_page_break()
  return self
```

```
if __name__ == '__main__':
    from pygments.styles import get_all_styles

word = Word.new(root=', page_break=False, font_size=7)
    for style in get_all_styles():
        word.add(_file__, style=style, title=style)
    word.save('demo.docx')
```

gruvbox-light

```
pygments = "^2.12.0"
python-docx = "^0.8.11"
import pathlib as p
import typing as t
from docx import Document
from docx.shared import Pt, RGBColor
from pygments.lexers import guess_lexer_for_filename
from pygments.styles import get_style_by_name
Path = t.Union[str, p.Path]
class Word:
  Self = __qualname__
  def __init__(
    root: Path = '.', style: str = 'default', page_break: bool = True,
    font_name: str = 'Times New Roman', font_size: int = 9,
    \underline{self}.\_root = p.Path(root).absolute()
     self._default = {
       'style': style,
       'page_break': page_break,
    self._doc = Document()
     font = \underline{self.\_doc.styles['Normal']}.font
     font.name = font_name
    font.size = Pt(font\_size)
  @classmethod
  def new(cls, *args, **kwargs) -> Self:
    return cls(*args, **kwargs)
  @property
  def root(self) -> p.Path:
    return self._root
  def add(
    path: Path, plain: bool = False,
    style: t.Optional[str] = None, page\_break: t.Optional[bool] = None,
    title: t.Optional[str] = None,
  ) -> Self:
    if plain:
       return self._add_plain(path, page_break, title)
     path = p.Path(path).absolute() \\
     code = path.read_text()
     lexer = guess\_lexer\_for\_filename(path.name, code)
```

```
styles = dict(get_style_by_name(style or self._default['style']))
     # heading
     self._doc.add_heading(title or path.relative_to(self._root).as_posix(), 0)
     # paragraph
     paragraph = self._doc.add_paragraph()
     for type, value in lexer.get_tokens(code):
       style = styles.get(type, {})
       run = paragraph.add\_run(value)
       # bold, italic, underline
       run.bold = style.get("bold", False")
       run.italic = style.get('italic', False)
       run.underline = style.get('underline', False)\\
       # color
       color = style.get(\c'color',\c'None)
       if color is not None:
         run.font.color.rgb = RGBColor.from\_string(color)
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
  def save(self, path: Path) -> Self:
     self._doc.save(path)
     return self
  def _add_plain(
     self,
     path: Path,
    page_break: t.Optional[bool] = None, title: t.Optional[bool] = None,
  ) -> Self:
     path = p.Path(path).absolute()
     \textcolor{red}{\textbf{self.\_doc.add\_heading(title or path.relative\_to(self.\_root).as\_posix(), 0)}}
     self._doc.add_paragraph(path.read_text())
     if page_break or self._default['page_break']:
       self._doc.add_page_break()
     return self
if __name__ == '__main__':
  from pygments.styles import get_all_styles
  word = Word.new(root='.', page\_break= \colored{False}, font\_size=7)
  for style in get_all_styles():
    word.add(__file__, style=style, title=style)
  word.save('demo.docx')
```