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,
    ) -> 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', 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')
```

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:
        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')
```

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,
    ) -> 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', 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')
```

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 = 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', 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')
```

autumn

```
'''
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', 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')
```

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']))
        # 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')
```

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,
    ) -> 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', 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')
```

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 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,
```

```python
    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', 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')
```

monokai

```
"""
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', 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')
```

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__
```

```python
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', 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')
```

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['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')
```

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['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')
```

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 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', 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')
```

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:
    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', 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')
```

fruity

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

from          import
from                    import
from                           import
from                           import




class
              __qualname__

    def __init__

                           '.'                   'default'                                 True
                                  'Times New Roman'                          9
           None


            'style'
            'page_break'


                                           'Normal'




    def new
        return


    def root
        return

    def add

                                              False
                                           None                                                   None
                                          None

        if
```

```
      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
```

```
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', 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')
```

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:
        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')
```

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 = __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', 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,
    ) -> 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', 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,
    ) -> 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', 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']))
        # 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')
```

igor

```
'''
pygments = "^2.12.0"
python-docx = "^0.8.11"
'''
import pathlib as p
import typing as t
```

```python
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', 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__(
        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', 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-dark

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

```python
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', 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')
```

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,
    ) -> 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', 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')
```

algol

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

```python
'''
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', 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')
```

algol_nu

```
'''
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', 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')
```

arduino

```
'''
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', 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')
```

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__(
        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', 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')
```

abap

```
'''
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', 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')
```

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,
```

```python
        }
        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')
```

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__

    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', 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')
```

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,
    ) -> 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', 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')
```

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] = 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')
```

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', 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
```

```python
    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

```python
'''
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', 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')
```

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__(
        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', 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')
```

zenburn

```
'''
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', 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')
```

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:
    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')
```

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__(
        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)
```

```python
        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')
```