

python-docx 使用教程

基本用法：创建、读取和保存文档

要开始使用 **python-docx**，首先需要创建或打开一个 Word 文档。通过 `Document()` 构造函数可以创建一个新的空白文档 ¹。例如：

```
from docx import Document

# 创建一个新的 Word 文档
document = Document()
```

如上，`Document()` 会基于默认模板创建一个空文档 ¹。如果要打开现有的 Word 文件，只需在调用 `Document()` 时传入文件路径即可，例如：`document = Document('已有文档.docx')`。²

完成文档内容的编辑后，可以使用 `Document.save(path)` 方法将文档保存到指定路径 ³：

```
document.save('输出文档.docx')
```

上面的代码将当前内存中的文档内容保存为 输出文档.docx 文件。

段落与标题

在 Word 文档中，**段落**是承载文本的基本块，标题也属于特殊的段落。使用 `Document.add_paragraph(text)` 方法可以在文档末尾添加一个段落，并可选择直接提供初始文本 ⁴：

```
doc = Document()
doc.add_paragraph('这是一个普通段落。')
```

如果需要添加**标题**，可以使用 `Document.add_heading(text, level)` 方法。`level` 参数指定标题级别，1 表示一级标题，2 表示二级标题，以此类推（最高可到 9）⁵。例如，添加一个一级标题和二级标题：

```
doc.add_heading('文档主标题', level=1) # 一级标题
doc.add_heading('次级标题示例', level=2) # 二级标题
```

`level=0` 则会添加一个“Title”样式的段落，通常用于文档题目 ⁶。段落添加后会返回一个 `Paragraph` 对象，可以进一步设置其格式。例如可以像下面这样插入段落并保存引用：

```
para = doc.add_paragraph('带格式的段落。')
```

然后通过 `para` 来调整格式。

段落对齐：可以通过段落对象的 `paragraph_format.alignment` 属性来设置段落的对齐方式。`python-docx` 提供了枚举值 `WD_ALIGN_PARAGRAPH` 表示对齐选项，包括左对齐、居中、右对齐、两端对齐等⁷。例如，将段落居中对齐：

```
from docx.enum.text import WD_ALIGN_PARAGRAPH

para = doc.add_paragraph('居中对齐段落')
para.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

行间距：可以通过 `paragraph_format.line_spacing` 属性来设置段落的行间距。既可以指定绝对值（例如 `Pt(18)` 表示18磅）也可以指定相对倍数（例如 1.5 表示1.5倍行距）⁸⁹。例如，将段落行距设置为1.5倍：

```
from docx.shared import Pt

para = doc.add_paragraph('行间距示例')
para.paragraph_format.line_spacing = 1.5 # 1.5倍行距
```

标题层级：使用 `add_heading` 添加的标题段落，其样式会自动设置为对应级别的样式（如“Heading 1”到“Heading 9”）。你也可以通过修改段落的 `style` 属性来调整已有段落为标题样式，如：`para.style = 'Heading 2'` 将一个段落设置为二级标题。



示例：使用 `python-docx` 添加一个标题（“GeeksForGeeks”，标题等级0相当于 Title 样式）和一个普通段落后的文档效果。如图所示，标题文字应用了标题样式（大号字体并带有下划线的格式），而下面的段落文本采用正文样式。

字符样式

在一个段落中，文字可以进一步细分为一个或多个 **运行 (Run)** 对象，每个运行是一段连续的字符，其可应用字符级别的样式。`python-docx` 提供了丰富的属性来设置运行的字体样式：

- **字体和字号**：通过获取运行的 `font` 属性（一个 `Font` 对象）来设置。如 `run.font.name = '宋体'` 可以将字体设为宋体，`run.font.size = Pt(18)` 将字号设为18磅¹⁰。需先从 `docx.shared` 导入 `Pt` 用于指定磅值。
- **粗体和斜体**：直接使用运行的布尔属性 `run.bold` 和 `run.italic`。将其赋值为 `True` 打开粗体或斜体，赋值 `False` 关闭。¹¹ 例如：

```
run = para.add_run('加粗文字')
run.bold = True
```

- **下划线**：通过运行的 `font.underline` 属性来控制。将其设为 `True` 会添加单下划线，`False` 为无下划线。如果不希望继承上层样式的下划线，也可将其设为 `None`。¹² 此外，`WD_UNDERLINE` 枚举提供了多种下划线样式（双线、虚线等）可供选择¹³。

```
from docx.enum.text import WD_UNDERLINE
run = para.add_run('下划线文字')
run.font.underline = WD_UNDERLINE.DASH # 虚线下划线
```

- **字体颜色**：通过 `run.font.color` 来设置颜色。¹⁴ 使用 `RGBColor` 可以指定RGB颜色值：

```
from docx.shared import RGBColor
run = para.add_run('彩色文字')
run.font.color.rgb = RGBColor(0xFF, 0x00, 0x00) # 红色
```

上例将文字设为红色（RGB 255,0,0）¹⁵。如果希望使用Word主题色，可以使用 `font.color.theme_color` 属性配合 `MSO_THEME_COLOR_INDEX` 枚举值。将颜色属性设为 `None` 则恢复自动（默认继承的颜色）¹⁶。

需要注意的是，很多字体属性（如粗体、斜体等）都是三态属性：`True` 表示启用，`False` 表示禁用，`None` 则表示“不设置”，由样式继承决定¹⁷¹⁸。例如一个运行的 `run.bold` 默认为 `None`，表示是否加粗取决于样式；如果显式设置为 `True/False` 则会覆盖样式的设定。

为了更好地组织文本，你也可以在添加段落时不直接给全部文字，然后通过多次 `add_run()` 来分别设置不同部分的格式。例如：

```
paragraph = doc.add_paragraph()
paragraph.add_run('普通文本，')
paragraph.add_run('加粗文本').bold = True
paragraph.add_run('，下划线文本').font.underline = True
```

这样最终段落会组合成“普通文本，加粗文本，下划线文本”，其中“加粗文本”为粗体，“下划线文本”带下划线。

表格

在自动化生成报告时，**表格**也是常用元素。使用 `Document.add_table(rows, cols)` 可以插入一个表格¹⁹。例如：

```
doc = Document()
table = doc.add_table(rows=2, cols=3)
```

上例将在文档末尾插入一个 2 行 3 列的表格¹⁹。`add_table` 返回一个 `Table` 对象。可以通过其属性和方法进行各种操作：

- **访问单元格**：使用 `table.cell(row_idx, col_idx)` 获取特定单元格（行列索引从0开始）²⁰。获取到的单元格是一个 `Cell` 对象，可对其 `text` 属性赋值来设置内容：

```
cell = table.cell(0, 1) # 第一行第二列
cell.text = '单元格内容'
```

或者通过表格的 `rows` 属性迭代行，再通过 `row.cells` 来访问每行中的各单元格²¹。例如：

```
for row in table.rows:
    for cell in row.cells:
        print(cell.text)
```

这样可以遍历表格中的所有单元格内容。

- **添加行列**：`table.add_row()` 方法可以在表格底部追加一行²²。类似地，`table.add_column()` 可以追加一列（但实际应用中较少需要手动添加列）。添加新行后，会得到新增行的 `Row` 对象或可以直接通过 `table.rows` 访问。常见用法是循环数据列表，逐行添加，例如：

```
data = [("苹果", 5), ("香蕉", 8), ("梨子", 3)]
table = doc.add_table(rows=1, cols=2)
# 设置表头
hdr_cells = table.rows[0].cells
hdr_cells[0].text = '水果'
hdr_cells[1].text = '数量'
# 添加数据行
for fruit, count in data:
    row_cells = table.add_row().cells
    row_cells[0].text = fruit
    row_cells[1].text = str(count)
```

- **单元格合并**：Word 允许将相邻的单元格合并。使用 `python-docx` 可以通过调用单元格对象的 `merge()` 方法实现²³。例如，将第一行前两列合并成一个单元格：

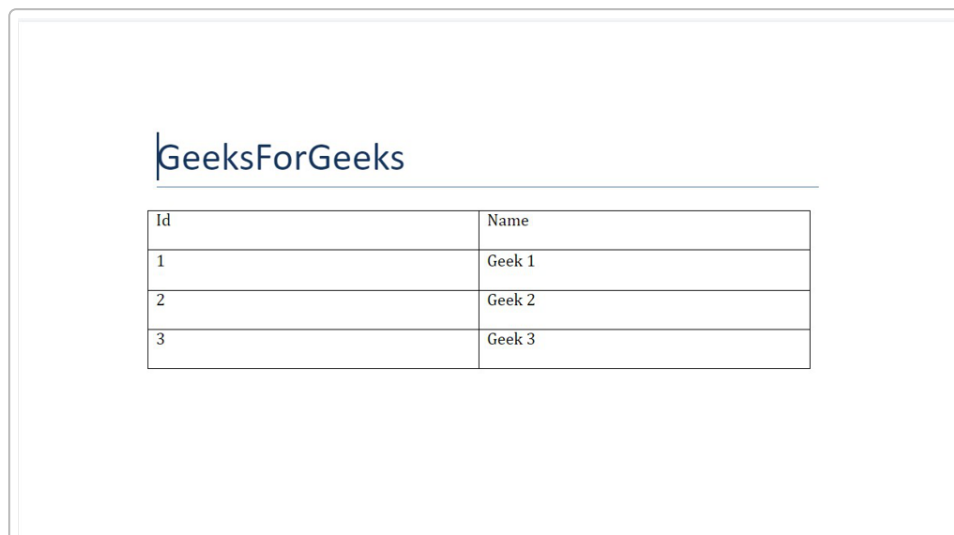
```
cell_00 = table.cell(0, 0)
cell_01 = table.cell(0, 1)
cell_00.merge(cell_01)
```

上述代码把表格第0行的第0和第1两个单元格合并成一个更宽的单元格。²⁴ 如果要纵向合并单元格，则提供同一列的不同行的单元格给 `merge()` 即可。**注意：**合并后原来的多个单元格内容会连在一起以段落分隔²⁵（如果有内容的话），未指定文本的单元格会被跳过。

- **表格样式：**可以应用 Word 内置的表格样式，使表格美观易读。通过设置 `table.style` 属性来达到此目的²⁶。样式名称可在 Word 中表格设计选项卡找到（比如“LightShading-Accent1”等）。例如：

```
table.style = 'Table Grid'
```

将给表格应用带网格线的标准表格样式（Table Grid）。内置样式名通常与Word界面显示的名称类似（注意去掉空格）。你也可以在Word中自定义样式然后在此处使用。



Id	Name
1	Geek 1
2	Geek 2
3	Geek 3

示例：使用 `python-docx` 创建的简单表格，包含表头“Id”和“Name”以及三行数据。可以看到表格采用默认样式，单元格边框为黑线矩形。通过设置 `table.style` 可以更改表格的外观样式。

图片

向文档中插入图片可以使用 `Document.add_picture(image_path, width=None, height=None)` 方法²⁷。此方法会在文档末尾创建一个段落，并将指定路径的图片插入其中²⁸。例如：

```
doc = Document()
doc.add_picture('image.png')
```

上面的代码会将当前目录下的 `image.png` 图片插入文档，图片将以其原始大小呈现²⁷。`add_picture` 可以选择可选参数 `width` 和 `height` 来指定图片尺寸²⁹。通常我们只需要指定其中一个，`python-docx` 会按比例缩放另一维以保持纵横比³⁰。例如，将图片宽度设为 1 英寸：

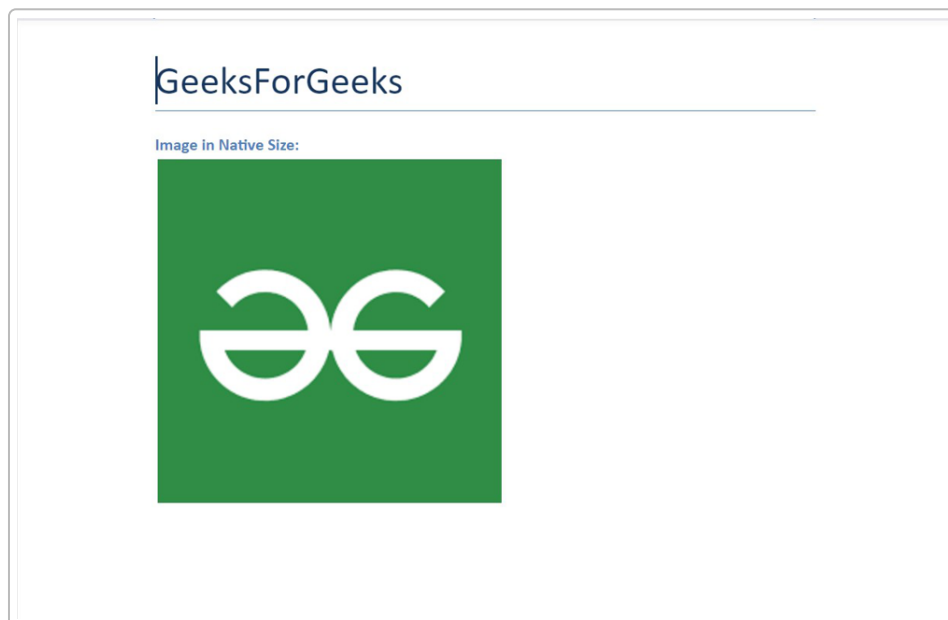
```
from docx.shared import Inches
doc.add_picture('image.png', width=Inches(1))
```

如果同时指定了宽度和高度，则图片会被强制拉伸到指定大小（可能会改变比例）。宽高参数可以使用 `docx.shared.Inches()` 或 `Cm()` 等辅助函数来提供更直观的度量单位。

插入图片后，该图片所在段落的对齐方式默认为左对齐。如果需要让图片居中或右对齐，可以通过调整其段落的对齐属性来实现。例如，让刚插入的图片居中：

```
last_paragraph = doc.paragraphs[-1] # 获取最后一个段落（即图片所在段落）
last_paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

上面代码将图片段落设置为居中对齐³¹。类似地，可以设置 `RIGHT` 实现右对齐。



示例：在文档中插入图片后的效果。上图插入了一张 GeeksForGeeks 的标志图片，并在标题下方显示。由于未指定 `width`/`height`，图片以原始尺寸插入，默认为左对齐。可以通过如上代码将其段落对齐方式改为居中或右对齐，或指定宽高调整图片大小。

样式设置

Word 提供了**样式**机制来统一管理文本的格式。python-docx 同样允许我们应用内置样式或自定义样式来格式化段落、文字和表格，以提高文档一致性。

- **内置样式的应用**：可以直接将段落的 `style` 属性设为某个样式名。例如：`paragraph.style = 'Heading 1'` 将段落应用一级标题样式³²；`paragraph.style = 'List Bullet'` 将段落应用项目符号列表样式³³。也可以在新增段落时通过参数指定样式，例如：`doc.add_paragraph('文本', style='Quote')` 会添加一个引用样式的段落。对于 `Run`（运行）对象，也有 `run.style` 属性，用来应用字符样式（例如 `'Emphasis'` 着重号样式等）³⁴。样式名通常使用英文，与 Word 界面显示名称相对应（即使你的 Office 为其它语言，也需使用其内部英文名称³⁵）。

- **查询样式：**通过 `document.styles` 可获取样式集合对象，然后可通过名称索引，例如 `document.styles['Normal']` 得到“Normal”（正文）样式对象³⁶。样式对象可以查看和修改其属性，比如段落样式的 `base_style`（基础样式）、`next_paragraph_style`（后续段落样式）等。
- **自定义样式：**可以新增自定义样式用于特殊格式需求。使用 `document.styles.add_style(name, WD_STYLE_TYPE)` 方法添加³⁷。例如，添加一种新的段落样式：

```
from docx.enum.style import WD_STYLE_TYPE
new_style = document.styles.add_style('MyStyle', WD_STYLE_TYPE.PARAGRAPH)
new_style.base_style = document.styles['Normal'] # 继承Normal样式
```

上述代码创建了名为“MyStyle”的段落样式，并以 Normal 为基础样式继承³⁸。接下来可以设置该样式的具体格式，例如：

```
new_style.font.name = 'Arial' # 字体
new_style.font.size = Pt(14) # 字号14磅
new_style.font.bold = True # 加粗
new_style.paragraph_format.space_before = Pt(12) # 段前间距12磅
new_style.paragraph_format.space_after = Pt(12) # 段后间距12磅
```

如此定义后，就可以通过 `para.style = 'MyStyle'` 来应用自定义样式了。注意，自定义样式名称应避免与内置样式重复，并且不会自动出现在 Word 样式库中，除非你设置其 `quick_style` 等属性使其可见³⁹⁴⁰。

- **表格样式：**表格对象的 `style` 属性同样可以设置为内置或自定义的表格样式名。例如 `table.style = 'LightShading-Accent1'`。²⁶

通过善用样式，我们可以一次性定义一组格式，然后在文档中反复使用，从而保证格式一致并简化代码。

页眉页脚与页码

页眉和页脚是每页顶部和底部的内容区域。使用 python-docx，可以通过文档的 `sections` 属性来访问每节的页眉页脚对象⁴¹。默认新建的文档只有一个节（`document.sections[0]`）。每个 `Section` 对象有 `header` 和 `footer` 属性，代表该节默认的页眉和页脚⁴²⁴³。例如，获取第一节的页眉：

```
section = document.sections[0]
header = section.header # <docx.section._Header 对象>
```

新文档开始时页眉是空的（未定义），此时 `header.is_linked_to_previous` 为 True，表示继承前一节（因为第一节没有前一节，所以无页眉显示）⁴⁴⁴⁵。一旦访问或添加了内容，python-docx 会自动创建页眉定义并将 `is_linked_to_previous` 设为 False⁴⁶。

向页眉或页脚添加内容的方法与向普通文档添加段落类似。每个 `_Header` 或 `_Footer` 对象本质上是一个内容容器，初始包含一个空段落⁴⁷⁴⁸。可以直接对其 `paragraphs[0]` 进行赋值，或用 `add_paragraph()` 添加新段落。例如，在页眉添加文本：

```
header.paragraphs[0].text = "机密文件 - 请勿外传"
# 或等效地：
# header.add_paragraph("机密文件 - 请勿外传")
```

这样就为当前节定义了页眉内容⁴⁹。同理可以操作 `section.footer` 来设置页脚内容。

如果希望首页页眉不同于后续页面（如封面不显示页眉），可以将 `section.different_first_page_header_footer` 设为 `True`，然后 `section.first_page_header` 会得到单独的页眉对象供首页使用⁵⁰。奇偶页不同的页眉也可通过类似方式实现（`odd_and_even_pages_header_footer`）。

页码插入：在 Word 中，页码通常通过插入域（Field）来实现，例如 `{ PAGE }` 表示当前页码。遗憾的是，截至目前版本，`python-docx` 尚不支持直接插入域来自动更新页码⁵¹。然而，可以通过操作底层 XML 手动插入一个页码域作为变通。⁵² 一般步骤是在页脚段落中添加一个运行，然后向运行的 XML 元素列表中依次追加表示域开始、域代码、域结束的元素。

例如，下面的示意性代码展示了如何在页脚添加当前页码域：

```
from docx.oxml import OxmlElement
from docx.oxml.ns import qn

footer = document.sections[0].footer
paragraph = footer.paragraphs[0]
run = paragraph.add_run() # 添加一个运行以容纳域

# 域开始
fld_char_begin = OxmlElement('w:fldChar')
fld_char_begin.set(qn('w:fldCharType'), 'begin')
# 域代码
instr_text = OxmlElement('w:instrText')
instr_text.set(qn('xml:space'), 'preserve')
instr_text.text = "PAGE" # 页码域代码
# 域结束
fld_char_end = OxmlElement('w:fldChar')
fld_char_end.set(qn('w:fldCharType'), 'end')

# 将上述XML元素按顺序添加到运行中
run._r.append(fld_char_begin)
run._r.append(instr_text)
run._r.append(fld_char_end)
```

上述代码在页脚段落插入了一个 `PAGE` 域。⁵³ ⁵⁴ 保存文档并打开后，Word 会显示实际页码。但可能需要手动更新域（一般在打开文档时会提示更新所有域）。

注意：使用域实现页码时，只有在 Word 中打开文档时才能看到动态页码。直接生成的 docx 文本中域显示为代码或可能为空白，需用户更新。对于复杂的页码格式（如“第 X 页 共 Y 页”），可以插入多个域（`PAGE` 和 `NUMPAGES`）并添加文字。

高级功能：目录、分节与页面布局

本节介绍一些高级操作，包括添加自动目录、分节控制以及页面布局设置等。

- **添加目录：**自动目录（Table of Contents）在 Word 中也是通过域来实现（一般是一个 TOC 域）。虽然 python-docx 没有高级API直接创建目录，但是可以采用与页码类似的方式插入一个 TOC 域来让 Word 自动生成目录。⁵⁵ ⁵⁶ 基本思路是在文档开头插入一个段落，添加一个运行，然后构造 TOC 域代码。示例：

```
paragraph = document.add_paragraph() # 占位段落，将成为目录
run = paragraph.add_run()
fld_begin = XmlElement('w:fldChar')
fld_begin.set(qn('w:fldCharType'), 'begin')
instr_text = XmlElement('w:instrText')
instr_text.set(qn('xml:space'), 'preserve')
instr_text.text = 'TOC \o "1-3" \h \z \u' # 目录域的指令代码 57
fld_separate = XmlElement('w:fldChar')
fld_separate.set(qn('w:fldCharType'), 'separate')
fld_end = XmlElement('w:fldChar')
fld_end.set(qn('w:fldCharType'), 'end')
for element in (fld_begin, instr_text, fld_separate, fld_end):
    run._r.append(element)
```

上述代码插入了一个“目录”域，设置包含1-3级标题、超链接且隐藏页码中的超链接下划线等开关（\o "1-3" \h \z \u 这些开关含义可参考Word域代码手册）⁵⁷。插入后首次打开文档时，Word 会提示更新新字段以生成目录内容⁵⁶。用户点击“是”后，Word 将根据文档中的标题生成目录及对应页码。⁵⁸ 如果用户跳过更新，则目录占位可能显示空白，需要后续手动更新。

提示：由于目录域需要Word客户端计算页码，纯靠 python-docx 无法预先计算。因此通常的做法是插入域后交给最终打开文档的用户/应用去更新。如果需要编程自动更新，可考虑结合 win32com 调用 Word 应用刷新域（超出本文范围）。

- **分节：**Word 文档可以有多个节，以便在不同部分使用不同的页面设置（如页眉页脚不同、页码重新编号、纸张方向不同等）。在 python-docx 中，通过 `Document.add_section(start_type)` 方法可以在文档末尾添加一个新节⁵⁹。`start_type` 参数使用 `WD_SECTION_START` 枚举，例如 `WD_SECTION_START.NEW_PAGE`（新页开始新节）、`ODD_PAGE`（下一个奇数页开始）等⁶⁰。每个新节继承上一节的大部分设置，例如页边距等，可以在创建后修改。

可以通过遍历 `document.sections` 来访问所有节⁶¹。每个 `Section` 对象提供了多种页面布局属性供读写，包括：

- **纸张方向：**`section.orientation`，取值为枚举 `WD_ORIENT.PORTRAIT`（纵向）或 `WD_ORIENT.LANDSCAPE`（横向）⁶²。改变 `orientation` 后通常需要交换宽高以生效⁶³。例如，将当前节切换为横向：

```
from docx.enum.section import WD_ORIENT
section = document.sections[-1]
section.orientation = WD_ORIENT.LANDSCAPE
# 交换宽高以应用横向
```

```
new_width, new_height = section.page_height, section.page_width
section.page_width = new_width
section.page_height = new_height
```

上述操作完成后，该节以及后续内容都会以横向页面布局显示 ⁶³。

- **纸张大小**：通过 `section.page_width` 和 `section.page_height` 来设置。需要以 EMU (English Metric Unit, 英文公制单位) 提供数值，可以借助 `Inches` 或 `Cm` 来转换。比如将纸张设置为 A4 大小：

```
section.page_width = Cm(21)    # A4宽度21厘米
section.page_height = Cm(29.7) # A4高度29.7厘米
```

纸张大小包括页边距区域 ⁶⁴。默认的新文档通常是 Letter 8.5x11英寸，可根据需要调整为A4等。

- **页边距**：`Section` 提供诸如 `left_margin`, `right_margin`, `top_margin`, `bottom_margin` 等属性来设置边距，返回值和设值均使用 `Length` 对象 ⁶⁵。例如：

```
from docx.shared import Inches
section.left_margin = Inches(1.5) # 左边距1.5英寸
section.right_margin = Inches(1)  # 右边距1英寸
section.top_margin = Cm(2)        # 上边距2厘米
section.bottom_margin = Cm(2)     # 下边距2厘米
```

设置页边距后，该节所有页面都会应用新的边距 ⁶⁵。还有 `header_distance` 和 `footer_distance` 属性可以设置页眉/页脚距页面边缘的距离（默认为0.5英寸） ⁶⁶。

通过分节，我们可以在同一文档中实现不同页面布局。例如封面无页眉页脚、正文有页眉页脚，某些页横向等。

以上就是 **python-docx** 的主要用法教程。通过本教程的各模块内容，具有一定 Python 基础的开发者应能掌握创建和修改 Word 文档的基本技巧，包括添加段落和标题、设置字体样式、构造表格和图片、应用样式，以及处理页眉页脚、分节和目录等高级功能。在实际应用中，可根据需要组合运用这些功能，自动化生成符合格式要求的 Word 报告文档。 ⁶⁷ ⁶⁸

¹ ⁴ ⁵ ⁶ ¹¹ ¹⁹ ²⁰ ²¹ ²² ²⁶ ²⁷ ²⁹ ³⁰ ³³ ³⁴ ⁶⁷ Quickstart — python-docx 1.2.0 documentation
<https://python-docx.readthedocs.io/en/latest/user/quickstart.html>

² ³ ²⁸ Document objects — python-docx 1.2.0 documentation
<https://python-docx.readthedocs.io/en/latest/api/document.html>

⁷ ⁸ ⁹ ¹⁰ ¹² ¹³ ¹⁴ ¹⁵ ¹⁶ ¹⁷ ¹⁸ Working with Text — python-docx 1.2.0 documentation
<https://python-docx.readthedocs.io/en/latest/user/text.html>

²³ ²⁴ ²⁵ Table - Merge Cells — python-docx 1.2.0 documentation
<https://python-docx.readthedocs.io/en/latest/dev/analysis/features/table/cell-merge.html>

31 Python docx Lib Center Align image - Stack Overflow

<https://stackoverflow.com/questions/26474551/python-docx-lib-center-align-image>

32 35 36 37 38 39 40 68 Working with Styles — python-docx 1.2.0 documentation

<https://python-docx.readthedocs.io/en/latest/user/styles-using.html>

41 44 45 46 47 49 Working with Headers and Footers — python-docx 1.2.0 documentation

<https://python-docx.readthedocs.io/en/latest/user/hdrftr.html>

42 43 48 50 62 64 66 Section objects — python-docx 1.2.0 documentation

<https://python-docx.readthedocs.io/en/latest/api/section.html>

51 Issue #1297 • python-openxml/python-docx - Page numbering - GitHub

<https://github.com/python-openxml/python-docx/issues/1297>

52 53 54 ms word - Add page number using python-docx - Stack Overflow

<https://stackoverflow.com/questions/56658872/add-page-number-using-python-docx>

55 56 57 58 Python: Create a "Table Of Contents" with python-docx/lxml - Stack Overflow

<https://stackoverflow.com/questions/18595864/python-create-a-table-of-contents-with-python-docx-lxml>

59 60 61 63 65 Working with Sections — python-docx 1.2.0 documentation

<https://python-docx.readthedocs.io/en/latest/user/sections.html>