副本-高效办公知识库

> 说明：用于之前已学“基础语法”、“高效办公”知识的功能概念、语法、示例的查阅。
> 操作说明：使用ctr+f（mac使用command+f），在搜索框中输入要查找的关键词进行查找，例如：os模块

目录一览

1 高效办公——openpyxl

1.1 openpyxl库

- openpyxl库可以处理Excel2010以后的电子表格格式，包括xlsx/xlsm/xltx/xltm格式的电子表格文档。
- openpyxl可以处理工作簿，工作表，单元格及单元格的值。
- openpyxl属于第三方库，因此需要下载安装才能使用。
  - Windows: *pip install openpyxl*
  - MacOS: *pip3 install openpyxl*

1.2 工作簿

1.2.2 打开工作簿 ： load_workbook()函数

- 功能
  - 打开**已有的工作簿**，返回工作簿对象（Workbook object）。
- 语法
  - *load_workbook(filename)*
    - 参数*filename*：工作簿的路径。
- 示例

```
from openpyxl import load_workbook
```

```
# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
```

1.2.3 创建工作簿：Workbook类

- 功能
  - 通过**实例化Workbook**类来创建工作簿对象。
- 语法
  - 实例化Workbook: ***Workbook()***
- 示例

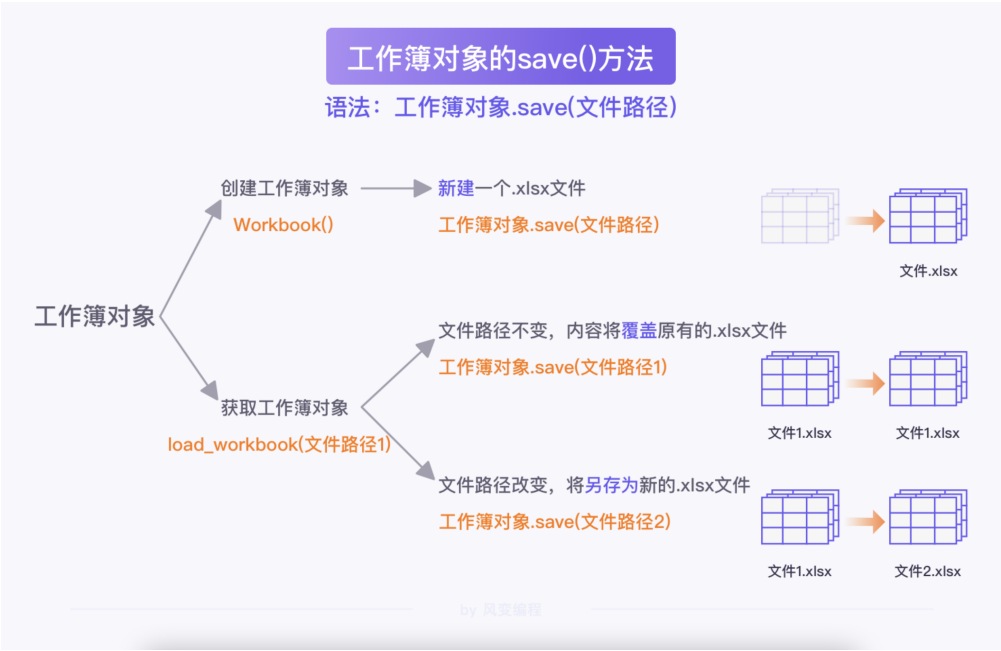
```
from openpyxl import Workbook

# 新建工作簿
new_wb = Workbook()
```

1.2.4 工作簿对象

- 概念
  - 根据openpyxl的定义，一个.xlsx格式的Excel文件就代表了一个工作簿对象（Workbook对象）。

1.2.5 保存工作簿：save()方法

- 功能
  - 保存工作簿到本地。



- 语法
  - ***工作簿对象.save(filename)***
    - 参数*filename*：新工作簿的文件路径。
- 示例

```
from openpyxl import Workbook

# 新建工作簿
```

```
new_wb = Workbook()
# 将新建的工作簿保存为【new_excel.xlsx】
new_wb.save('./new_excel.xlsx')
```

1.2.6 获取活动工作表：active属性

- 功能
  - 获取工作簿中的**活动工作表。**
- 语法
  - ***工作簿对象.active***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active
```

1.3 工作表

1.3.1 获取指定工作表：工作簿对象[表名]

- 功能
  - 如果我们已知工作表的名称，就可以用**表名为索引**，获取到指定的工作表对象。
- 语法
  - ***工作簿对象['表名']***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 按表名取表
ws1 = wb['sheet1']
ws2 = wb['sheet2']
```

1.3.2 工作表对象

- 概念
  - 工作表就是工作簿中实际显示的工作表页面，其名称显示在下方的标签上。
  - 现实中的工作表，在openpyxl中对应着**工作表对象（Worksheet object）**。

1.3.3 获取单行：工作表对象[行数]

- 功能
  - 获取工作表中的**单行**。
  - 返回一个包含该行所有单元格对象的元组。
- 语法
  - ***工作表对象[行数]***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# 获取第五行
row = ws[5]
```

1.3.4 获取单列：工作表对象['列名']

- 功能
  - 获取工作表中的**单列**。
  - 返回一个包含该列所有单元格对象的元组。
- 语法
  - ***工作表对象['列名']***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# 获取第二(B)列
col = ws['B']
```

1.3.5 获取多行：iter_rows()方法

- 功能
  - 获取表格中指定范围内的**多行**。
  - 返回一个**可迭代对象**，该对象中有n个元组，n为参数中指定的行数，每一个元组都代表了表格中的一行。
  - 通常情况下会和for循环结合使用，从而使得我们取出其返回的可迭代对象中的每一个元组，即表格中指定范围内的每一行数据。
- 语法
  - ***工作表对象.iter_rows(min_row, max_row, min_col, max_col, values_only=False)***
    - 参数*min_row*和*max_row*：分别表示指定范围的**最小行索引**和**最大行索引**。
    - 参数*min_col*和*max_col*：分别表示指定范围的**最小列索引**和**最大列索引**。
    - 参数*values_only*：决定返回单元格对象还是单元格的值，如果为默认的False就返回**单元格对象**，为True则返回**单元格的值**。
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
```

```
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# 返回第2行至第12行，第2(B)列值第3(C)列这个范围内的所有数据
for row in ws.iter_rows(min_row=2, max_row=12,
                        min_col=2, max_col=3, values_only=True):
    print(row)
```

1.3.6 添加一行：append()方法

- 功能
  - 将部分可迭代对象（常见的如**列表**，**元组**）添加到工作表对象中，即给表格的末尾追加一行数据。
- 语法
  - ***工作表对象.append(列表/元组)***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

info_tuple = ('S1911', '方达仁', 20000, '产品')

# 将info_tuple添加到工作表
ws.append(info_tuple)
```

1.4 单元格

1.4.1 获取单元格

- **for row in 工作表对象.iter_rows()**
  - 功能
    - 获取指定范围的行，当参数*values_only*为默认的False时，得到的row就是一个个由单元格对象组成的元组，可以通过索引或者for循环遍历的方式来获取单独的单元格对象。
  - 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

```

```
# 返回第2行至第12行，第2(B)列值第3(C)列这个范围内的所有单元格对象
for row in ws.iter_rows(min_row=2, max_row=12,
                        min_col=2, max_col=3):
    for cell in row:
        print(cell)
```

- **for cell in 工作表对象[行数]**
  - 功能
    - 通过行数来指定具体的行，然后通过for循环遍历获取指定行中的每一个单元格对象。
  - 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# for循环遍历，取出第三行的所有单元格对象
for cell in ws[3]:
    print(cell)
```

- **for cell in 工作表对象['列名']**
  - 功能
    - 通过列名来指定具体的列，然后通过for循环遍历获取指定列中的每一个单元格对象。
  - 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# for循环遍历，取出第三(C)列的所有单元格对象
for cell in ws['C']:
    print(cell)
```

- **工作表对象['单元格坐标']**
  - 功能
    - 直接通过单元格的坐标来获取具体的单元格对象。
  - 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
```

```
ws = wb.active

# 打印单元格对象A1
print(ws['A1'])
```

1.4.2 单元格对象

- 概念
  - 单元格对象代表工作表中的一个单元格。

1.4.3 单元格取值与赋值：value属性

- 功能
  - 获取单元格的具体数据。
  - 通过给单元格对象.value属性重新赋值的方式，修改单元格的值或给单元格添加值。
- 语法
  - 获取单元格的值
    - ***单元格对象.value***
  - 给单元格对象赋值
    - ***单元格对象.value = 值***
- 示例

```
from openpyxl import load_workbook

# 打开【demo_excel.xlsx】工作簿
wb = load_workbook('./demo_excel.xlsx')
# 获取活动工作表
ws = wb.active

# 打印单元格对象A1的值
print(ws['A1'].value)

# 修改单元格对象C2的值为1000
ws['C2'].value = 1000
```

1.5 样式

1.5.1 调整列宽

- 语法
  - 工作表对象.column_dimensions[’列名’].width = 列宽
    - 列宽可以为**整数**

1.5.2 定义单元格样式

- 功能
  - **填充颜色**、**边框**、**对齐方式**，分别对应单元格对象的*fill*、*border*、*alignment*三个属性。

    而这三个属性需要的值，分别是*PatternFill*、*Border*、*Alignment*对象。

- 通过实例化*PatternFill*、*Border*、*Alignment*三个类来设置填充颜色、边框、对齐方式，然后将实例化后的对象赋值给单元格对象的*fill*、*border*、*alignment*三个属性。

- 示例
  - **设置填充颜色**

```
# 定义表头颜色样式为橙色
header_fill = PatternFill('solid', fgColor='FF7F24')

# 设置单元格填充颜色
cell.fill = header_fill
```

  - **设置边框**

```
# 定义边样式为细条
side = Side('thin')
# 定义表头边框样式，有底边和右边
header_border = Border(bottom=side, right=side)
# 定义表中、表尾边框样式，有左边
content_border = Border(left=side)

# 设置单元格边框
cell1.border = header_border
cell2.border = content_border
```

  - **设置对齐方式**

```
# 定义对齐样式横向居中、纵向居中
align = Alignment(horizontal='center', vertical='center')

# 设置单元格对齐方式
cell.alignment = align
```

2 高效办公——简单功能块

2.1 对文件/文件夹的操作

2.1.1 获取所有文件名

```
import os

# 设置文件夹路径
path = '文件夹路径'
# 获取文件夹内所有文件名和文件夹名
file_names = os.listdir(path)

```

```
# 循环取出每一个文件(夹)名
for file_name in file_names:
    print(file_name)
```

2.1.2 打开工作簿，获取活动工作表

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active
```

2.1.3 打开工作簿，获取指定工作表

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 按表名获取工作表
ws = wb['工作表名']
```

2.1.4 新建工作簿，打开工作表

```
from openpyxl import Workbook

# 新建工作簿
wb = Workbook()
# 获取工作表
ws = wb.active
```

2.1.5 打开文件夹下所有工作簿和其工作表

```
import os
from openpyxl import load_workbook

# 设置文件夹路径
path = '文件夹路径'
# 获取文件夹内所有文件名和文件夹名
file_names = os.listdir(path)

# 循环打开文件夹下所有工作簿和其工作表
for filename in file_names:
    wb = load_workbook(path + filename)
    ws = wb.active
```

2.1.6 保存工作簿

```
wb.save('工作簿路径')
```

2.2 对表格进行读取操作

2.2.1 读取单元格

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 读取单元格的值
cell_value = ws['单元格坐标'].value
```

2.2.2 读取行

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 按行读取
for row in ws.iter_rows(min_row=2, values_only=True):
    # 打印每一行的值
    print(row)
```

2.2.3 读取表头

```
froml openpyxl importoad_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 第一行为表头，循环输出表头的值
```

```
for cell in ws[1]:
    print(cell.value)
```

2.2.4 读取行内数据

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 按行读取
for row in ws.iter_rows(min_row=2, values_only=True):
    # 读取并打印行内单元格的值
    print(row[索引1])
```

2.2.5 将行内数据存入字典

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 创建字典
info_dict = {}

# 按行读取
for row in ws.iter_rows(min_row=2, values_only=True):
    # 取出 row 中索引为 1 的值
    dict_key = row[索引1]
    # 取出 row 中索引为 2 的值
    dict_value = row[索引2]
    info_dict[dict_key] = dict_value
```

2.3 对表格进行写入操作

2.3.1 写入单元格

```
from openpyxl import load_workbook

# 设置工作簿路径
```

```
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 给单元格赋值
ws['单元格坐标'].value = 值

# 保存工作簿
wb.save('工作簿路径')
```

2.3.2 写入单行

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 创建列表(也可以使用元组)保存单行数据
info_list = [1, 2, 3, 4, 5]

# 写入单行数据
ws.append(info_list)

# 保存工作簿
wb.save('工作簿路径')
```

2.3.3 写入行内数据

```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active

# 按行写入
for row in ws.iter_rows(min_row=2):
    # 将值写入本行对应坐标位置
    row[索引].value = 值

```

```
# 保存工作簿
wb.save('工作簿路径')
```

2.4 数据筛选

2.4.1 筛选出满足条件的数据

- 构造筛选条件

构造筛选条件的常见Python基础语法

Python基础语法	筛选条件	示例
条件判断	条件	If 条件
比较运算	条件：A等于B	If A == B
	条件：A不等于B	If A != B
	条件：A大于B	If A > B
	条件：A小于B	If A < B
	条件：A大于等于B	If A >= B
	条件：A小于等于B	If A <= B
成员运算	条件：A在指定的序列B中	If A in B
	条件：A不在指定的序列B中	If A not in B
逻辑运算	条件1与条件2同时成立	If 条件1 and 条件2
	条件1成立或条件2成立	If 条件1 or 条件2
	不满足条件1	If not 条件1


```
from openpyxl import load_workbook

# 设置工作簿路径
path = '工作簿路径'
# 打开工作簿
wb = load_workbook(path)
# 获取活动工作表
ws = wb.active


```

```
# 按行获取数据
for row in ws.iter_rows(min_row=2, values_only=True):
    # 获取数据a
    a = row[索引1]
    # 获取数据b
    b = row[索引2]

    # 构造筛选条件，如a满足条件1并且b满足条件2
    if a == 条件1 and b == 条件2:

```

2.5 数据匹配

2.5.1 匹配两表中有联系的数据

```
from openpyxl import load_workbook

# a为工作簿1
a_wb = load_workbook('工作簿路径1')
a_ws = a_wb.active

# b为工作簿2
b_wb =load.workbook('工作簿路径2')
b_ws = b_wb.active

info_dict = {}

# 循环读取出表头外的表格数据
for a_row in a_ws.iter_rows(min_row=2, values_only=True):
    # 取出 a_row 中索引为 1 的值
    a_key = a_row[索引1]
    # 取出 a_row 中索引为 2 的值
    a_value = a_row[索引2]
    # 键值对写入字典info_dict
    info_dict[a_key] = a_value

# 循环读取出表头外的表格数据
for b_row in b_ws.iter_rows(min_row=1, values_only=True):
    # 取出 b_row 中索引为 3 的值
    b_key = b_row[索引3]
    # 数据匹配，（也可以用 info_dict[b_key]）
    info_dict.get(b_key)
```

3 高效办公—— 邮件

3.1 smtplib 模块

3.1.1 设置邮箱服务器的端口信息

- 语法
  - SMTP_SSL(host, port)
    - 参数 host：服务器地址
    - 参数 port：端口号
- 示例

```
import smtplib

# 设置邮箱服务器，端口
smtp = smtplib.SMTP_SSL('smtp.qq.com', 465)
```

3.1.2 登录邮箱

- 语法
  - login(邮箱地址, 邮箱授权码)
    - 参数 **邮箱地址**：一般指邮箱账户
    - 参数 **邮箱授权码**：一般指邮箱授权码
- 示例

```
import smtplib

# 设置邮箱账号
account = input('请输入邮箱账户：')
# 设置邮箱授权码
token = input('请输入邮箱授权码：')
# 设置邮箱服务器，端口
smtp = smtplib.SMTP_SSL('smtp.qq.com', 465)
# 登录qq邮箱
smtp.login(account, token)
```

3.1.3 发送邮件

- 语法
  - sendmail(from_addr, to_addrs, msg)
    - 参数 from_addr：发件邮箱地址
    - 参数 to_addrs：收件邮箱地址
    - 参数 msg：邮件内容，需要注意的是，要调用as_string()转化为字符串
- 示例

```
# 发送邮件
smtp.sendmail(account, 'example@mail.com', msg.as_string())
```

3.1.4 关闭邮箱服务

- 语法
  - quit()
- 示例

```
# 关闭邮箱服务
smtp.quit()
```

3.2 email 模块 —— MIME模块

3.2.1 添加正文

- 语法
  - MIMEText(_text, _subtype, _charset)
    - 参数 msg：文本内容
    - 参数 type：类型，默认为 plain
    - 参数 charset：编码，一般为 utf-8
- 示例

```
from email.mime.text import MIMEText

content = '这是一份文字内容的邮件'
# 创建简单邮件对象
email_content = MIMEText(content, 'plain', 'utf-8')
```

3.2.2 添加附件

- 语法
  - MIMEText(_text, _subtype, _charset)
    - 参数 msg：附件内容
    - 参数 type：类型，附件内容一般为 base64
    - 参数 chartset：编码，一般为 utf-8
  - add_header('Content-Disposition', 'attachment', filename)
- 示例

```
# 设置内容类型为附件
attachment = MIMEText(file_data, 'base64', 'utf-8')
# 设置附件标题以及文件类型
attachment.add_header('Content-Disposition', 'attachment', filename='04_月考勤表.xlsx')
```

3.2.4 添加简单邮件对象到复合邮件对象中

- 语法
  - attach()
- 示例

```
from email.mime.multipart import MIMEMultipart

# 创建复合邮件对象
msg = MIMEMultipart()
```

```
# 添加正文到复合邮件对象中
msg.attach(email_content)
# 添加附件到复合邮件对象里
msg.attach(attachment)
```

3.2.5 设置邮件信息

- 语法
  - 设置发件人：MIMEMultipart()['From']
  - 设置收件人：MIMEMultipart()['To']
  - 设置主题：MIMEMultipart()['Subject']

4 高效办公——docx

4.1 docx 库

- docx 库可以用于创建和处理 word 文档。它的功能非常强大，它会把 word 中的文档、段落、文本等作为对象进行处理，可以创建和打开文件 word 文档、增加标题、改变行间距、调整对齐方式、增添段落、改变字体等等。
- docx 库只能处理 docx 文件，无法处理 doc 文件。
- 使用 docx 库需要安装 python-docx 库，该库属于第三方库，因此需要下载安装才能使用。
  - Windows 系统：pip install python-docx
  - Mac 系统：pip3 install python-docx

4.2 Word 文件

4.2.1 打开 Word 文件

- 功能
  - 打开**已有的 Word 文件**，返回 Document 对象。
- 语法
  - *Document(docx)*
    - 参数 ***docx***：Word 文件的路径。
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')
```

4.2.2 Document 对象

- 概念
  - 根据 python-docx 库的定义，一个.docx 格式的 Word 文件就代表了一个 Document 对象。

4.2.3 创建 Word 文件

- 功能
  - 通过实例化 Document 类来创建 Document 对象。
- 语法
  - 实例化 Document() 类：Document()
- 示例

```
from docx import Document

# 新建 Word 文件
new_doc = Document()
```

4.2.4 保存 Word 文件

- 功能
  - 保存 Word 文件到本地。
- 语法
  - *Document 对象.save(path)*
    - 参数 *path*：新 Word 文件的文件路径。
- 示例

```
from docx import Document

# 新建 Word 文件
new_doc = Document()
# 将新建的 Word 文件保存为【new_word.docx】
new_doc.save('./new_word.docx')
```

4.3 文档段落

4.3.1 Paragraph 对象

- 概念
  - Paragraph 对象是最频繁使用的块级元素，对应 Word 文件中的段落。

4.3.2 查看 Paragraph 对象

- 功能
  - 返回一个列表，列表中的每一个元素都是一个 Paragraph 对象。
- 语法
  - *Document 对象.paragraphs*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 循环遍历 Document 对象中的每一个 Paragraph 对象
```

```
for para in doc.paragraphs:
    # 打印 Paragraph 对象
    print(para)

# 打印 Paragraph 对象的个数
print(len(doc.paragraphs))
```

4.3.3 查看 Paragraph 对象中的文字

- 功能
  - 返回段落中包含的文字。
- 语法
  - *Paragraph 对象.text*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 循环遍历 Document 对象中的每一个 Paragraph 对象
for para in doc.paragraphs:
    # 打印 Paragraph 对象中的文字
    print(para.text)
```

4.3.4 添加 Paragraph 对象：Document 对象.add_paragraph()

- 功能
  - 对应 Word 中的段落，在 Word 文档里，将“Enter”回车键作为段落之间的分界点，每按一次回车，光标进入下一段，就意味着多了一个段落。
- 语法
  - *Document 对象.add_paragraph(text)*
    - **参数 text：直接在生成的 Paragraph 对象中添加文字，可以不写**
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()
```

4.4 段落中不同样式的文本

4.4.1 Run 对象

- 概念
  - Run 对象是段落中具有相似属性（如相似的字体大小，字体形状和字体样式）的连续单词。

4.4.2 查看 Run 对象

- 功能
  - 返回一个列表，列表中的每一个元素都是一个 Run 对象。
- 语法
  - *Paragraph 对象.runs*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 循环遍历 Document 对象中的每一个 Paragraph 对象
for para in doc.paragraphs:
    # 循环遍历 Paragraph 对象中的每一个 Run 对象
    for run in para.runs:
        # 打印 Run 对象
        print(run)
        # 打印 Run 对象
        print(run)
```

4.4.3 查看 Run 对象中的文字

- 功能
  - 返回 Run 对象中包含的文字。
- 语法
  - *Run 对象.text*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 循环遍历 Document 对象中的每一个 Paragraph 对象
for para in doc.paragraphs:
    # 循环遍历 Paragraph 对象中的每一个 Run 对象
    for run in para.runs:
        # 打印 Run 对象
        print(run)
        # 打印 Run 对象中的文字
        print(run.text)
```

4.4.4 添加 Run 对象

- 功能
  - 给 Paragraph 对象添加 Run 对象
- 语法
  - *Paragraph 对象.add_run(text)*
    - *参数 text：直接给生成的 Run 对象中添加文字，可以不写*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()

# 添加 Run 对象
run = doc.add_run()
```

4.4.5 给 Run 对象添加文字

- 功能
  - 给 Run 对象添加文字
- 语法
  - *Run 对象.add_text(text)*
    - *参数 text : Run 对象中需要添加的文字，一般为字符串类型*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()

# 添加 Run 对象
run = doc.add_run()

# 添加文字
run.add_text("天地不仁，以万物为刍狗；圣人不仁，以百姓为刍狗。")
```

4.4.6 给 Run 对象添加图片

- 功能
  - 给 Run 对象添加图片

- 语法
  - *Run 对象.add_picture(图片路径)*
    - *参数 图片路径 : 表示需要添加图片的路径*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()

# 添加 Run 对象
run = doc.add_run()

# 添加图片
run.add_picture("./图库/老子.png")
```

4.4.7 添加 Run 对象的同时添加文字

- 功能
  - 给 Paragraph 对象添加 Run 对象的同时添加文字
- 语法
  - *Paragraph 对象.add_run(text)*
    - *参数 text : Run 对象中需要添加的文字，一般为字符串类型*
- 示例

```
from docx import Document

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()

# 添加 Run 对象
run = doc.add_run("天地不仁，以万物为刍狗；圣人不仁，以百姓为刍狗。")
```

4.5 设置样式

4.5.1 设置段落样式：Paragraph 对象的属性 paragraph_format.alignment

- 功能
  - 控制段落的样式，即段落在页面的位置，例如对齐方式、缩进和段落前后的间距。
- 语法
  - *设置对齐方式：paragraph_format.alignment = 对齐方式*

- ***参数 对齐方式：***

4 种对齐方式

from docx.enum.text import WD_ALIGN_PARAGRAPH	
左对齐（默认）	Paragraph 对象.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.LEFT
右对齐	Paragraph 对象.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.RIGHT
居中对齐	Paragraph 对象.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
两边对齐	Paragraph 对象.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY

by 风变编程

- 示例

```
from docx import Document
from docx.enum.text import WD_ALIGN_PARAGRAPH

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()

# 设置对齐方式为左对齐
para.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.LEFT
```

4.5.2 设置字体样式：Run 对象的属性 font

- 功能
  - 该属性可以设置字体相关功能，包括斜体、加粗、大小、颜色等。
- 语法
  - ***设置字体大小：Run 对象.font.size = Pt(字体大小)***
  - ***设置加粗：Run 对象.font.bold = True***

- ***参数：***

font 补充

from docx.shared import RGBColor, Pt		
参数	说明	示例
指定字体	Run 对象.font.name	font.name = 'Calibri'
设置斜体	Run 对象.font.italic	font.italic = True
设置下划线	Run 对象.font.underline	font.underline = True
设置字体颜色	Run 对象.font.color.rgb	font.color.rgb = RGBColor(255, 0, 0)

- 示例

```
from docx import Document
from docx.shared import Pt

# 打开【demo_word.docx】Word 文件
doc = Document('./demo_word.docx')

# 添加 Paragraph 对象
para = doc.add_paragraph()


# 添加 Run 对象
run = doc.add_run("天地不仁，以万物为刍狗；圣人不仁，以百姓为刍狗)

# 设置字体大小为 14pt
run.font.size = Pt(14)

# 设置字体加粗
run.font.bold = True
```

5 高效办公——os

5.1 os模块

- os——Operating System（操作系统），它属于Python自带的标准库，不需要下载。
- 常用于处理文件和目录（文件夹）的操作，例如删除文件，判断文件目录是否存在等功能。

5.1.1 获取文件和文件夹名：listdir()函数

- 功能
  - 获取文件夹下的所有**文件名称**和**文件夹名称**。

    - 返回一个列表，包含指定的文件夹内的文件或文件夹的名称。
- 语法
    - *os.listdir(path)*
        - 参数*path*：文件夹路径
- 示例

```
import os

path = '文件夹路径'
filenames = os.listdir(path)
```

6 基础语法

基础语法知识库：https://docs.forchange.cn/docs/zdkyBjyGvoUybwA6