

子集和超级集

集合可以是其他集合的子集或超集:

- 子集: `issubset()`
- 超级集: `issuperset()`

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item2', 'item3'}
st2.issubset(st1) # True
st1.issuperset(st2) # True
```

例:

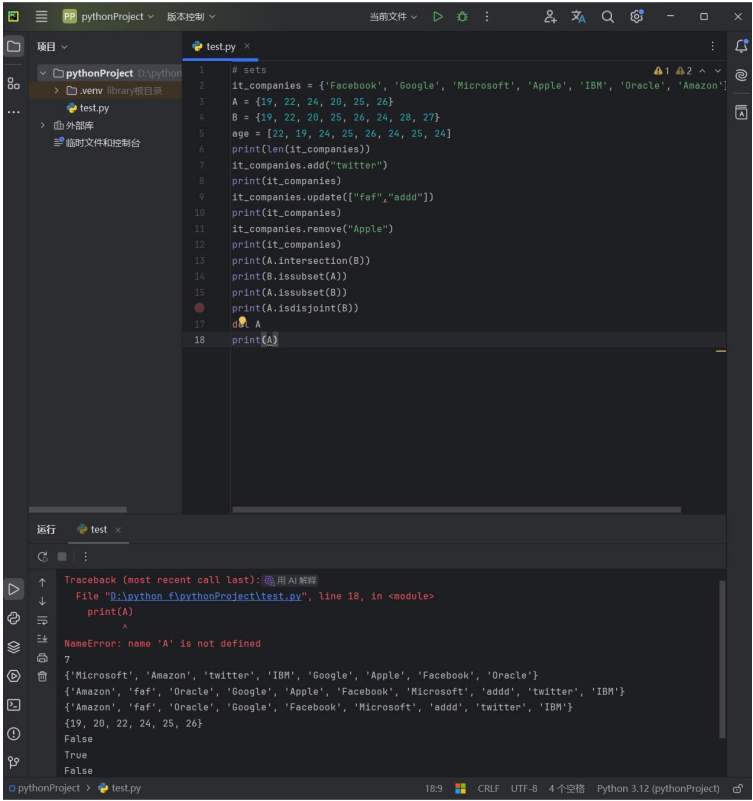
```
whole_numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
even_numbers = {0, 2, 4, 6, 8, 10}
whole_numbers.issubset(even_numbers) # False, because it is a super s
whole_numbers.issuperset(even_numbers) # True

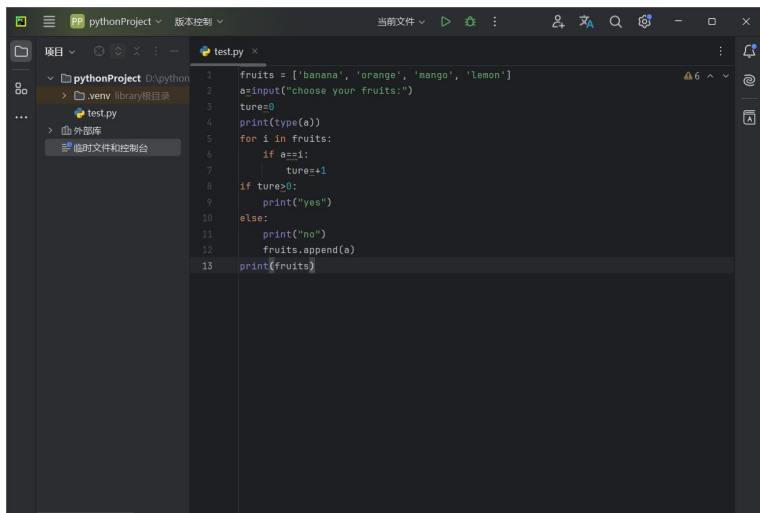
python = {'p', 'y', 't', 'h', 'o', 'n'}
dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
python.issubset(dragon) # False
```

1. 求集it_companies的长度
2. 在it_companies添加“Twitter”
3. 一次将多个 IT 公司插入到设置的it_companies
4. 从设置的it_companies中删除其中一家公司
5. remove 和 discard 有什么区别

练习：2级

1. 加入 A 和 B
2. 查找 A 交叉点 B
3. 是 B 的 A 子集
4. A 和 B 是不相交的集合吗
5. 将 A 与 B 连接，将 B 与 A 连接
6. A 和 B 的对称区别是什么
7. 完全删除集





匿名函数

如下图代码，我们可以：

- 通过def关键字，定义一个函数，并传入，如下图：

```
def test_func(compute):
    result = compute(1, 2)
    print(result)

def compute(x, y):
    return x + y

test_func(compute) # 结果: 3
```

- 也可以通过lambda关键字，传入一个一次性使用的lambda匿名函数

```
def test_func(compute):
    result = compute(1, 2)
    print(result)

test_func(lambda x, y: x + y) # 结果: 3
```

打开文件

open()打开函数

在Python，使用open函数，可以打开一个已经存在的文件，或者创建一个新文件，语法如下

```
open(name, mode, encoding)
```

name: 是要打开的目标文件名的字符串(可以包含文件所在的具体路径)。

mode: 设置打开文件的模式(访问模式): 只读、写入、追加等。

encoding: 编码格式 (推荐使用UTF-8)

示例代码:

```
f = open('python.txt', 'r', encoding=" UTF-8")
# encoding的顺序不是第三位，所以不能用位置参数，用关键字参数直接指定
```

注意事项

注意：此时的`f`是`open`函数的文件对象，对象是Python中一种特殊的数据类型，拥有属性和方法，可以使用对象.属性或对象.方法对其进行访问，后续面向对象课程会给大家进行详细的介绍。

mode常用的二种基础访问模式

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑， 原有内容会被删除 。如果该文件 不存在 ， 创建新文件 。
a	打开一个文件用于追加。如果该文件已存在， 新的内容将会被写入到已有内容之后 。如果该文件 不存在 ， 创建新文件 进行写入。

Read 读取会有不连续性质

read() 方法:

文件对象.read(num)

num表示要从文件中读取的数据的长度（单位是字节），如果没有传入num，那么就表示读取文件中所有的数据。

readlines() 方法:

readlines可以按照行的方式把整个文件中的内容进行一次性读取，并且返回的是一个列表，其中每一行的数据为一个元素。

```
f = open('python.txt')
content = f.readlines()

# ['hello world\n', 'abcdefg\n', 'aaa\n', 'bbb\n', 'ccc']
print(content)

# 关闭文件
f.close()
```

readline() 方法: 一次读取一行内容

```
f = open('python.txt')

content = f.readline()
print(f'第一行: {content}')

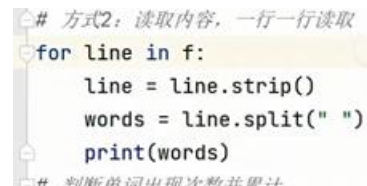
content = f.readline()
print(f'第二行: {content}')

# 关闭文件
f.close()
```

for循环读取文件行

```
for line in open("python.txt", "r"):
    print(line)
```

每一个line临时变量，就记录了文件的一行数据

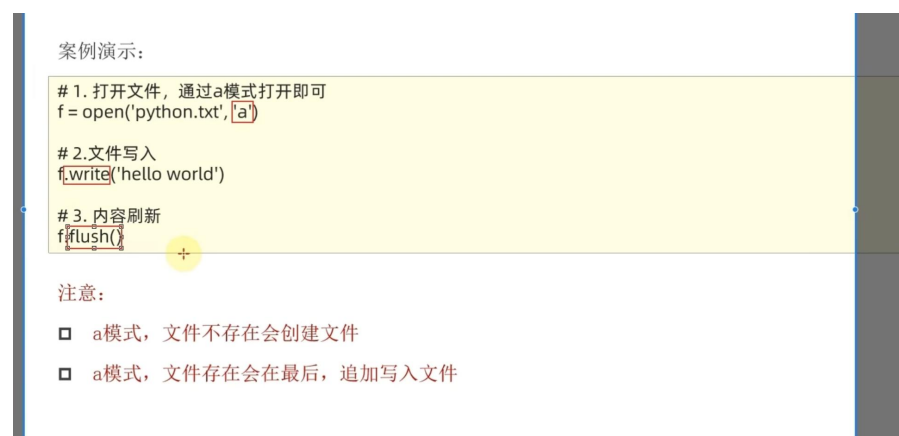


写操作 W

文件不存在会创建，文件存在会清空后再写入



Close 内置 flush 功能



```

# 打开文件得到文件对象, 准备读取
fr = open("D:/bill.txt", "r", encoding="UTF-8")
# 打开文件得到文件对象, 准备写入
fw = open("D:/bill.txt.bak", "w", encoding="UTF-8")
# for循环读取文件
for line in fr:
    line = line.strip()
    # 判断内容, 将满足的内容写出
    if line.split(",")[4] == "测试":
        continue # continue进入下一次循环, 这一次后面的内容就跳过了
    # 将内容写出去
    fw.write(line)
    # 由于前面对内容进行了strip()的操作, 所以要手动写出换行符
    fw.write("\n")

# close2个文件对象

# 打开文件得到文件对象, 准备读取
fr = open("D:/bill.txt", "r", encoding="UTF-8")
# 打开文件得到文件对象, 准备写入
fw = open("D:/bill.txt.bak", "w", encoding="UTF-8")
# for循环读取文件
for line in fr:
    line = line.strip()
    # 判断内容, 将满足的内容写出
    if line.split(",")[4] == "测试":
        continue # continue进入下一次循环, 这一次后面的内容就跳过了
    # 将内容写出去
    fw.write(line)
    # 由于前面对内容进行了strip()的操作, 所以要手动写出换行符
    fw.write("\n")

# close2个文件对象

```

异常捕获

基本语法:

```

try:
    可能发生错误的代码
except:
    如果出现异常执行的代码

```

快速入门

需求: 尝试以`r`模式打开文件, 如果文件不存在, 则以`w`方式打开。

```

try:
    f = open('linux.txt', 'r')
except:
    f = open('linux.txt', 'w')

```

捕获指定异常

基本语法:

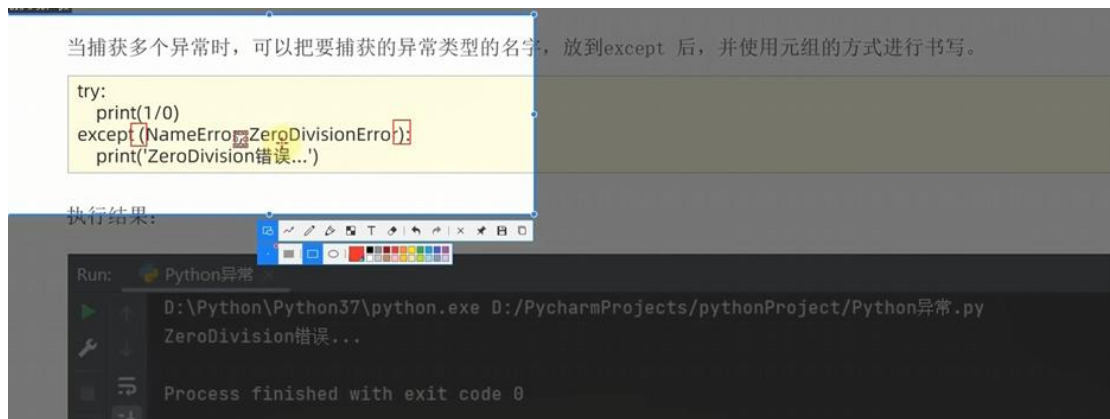
```

try:
    print(name)
except NameError as e:
    print('name变量名称未定义错误!')

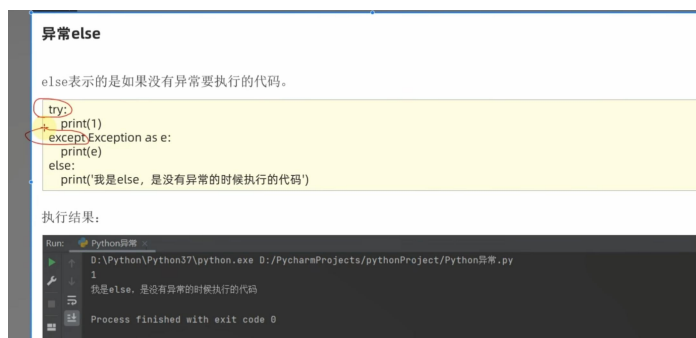
```

注意事项

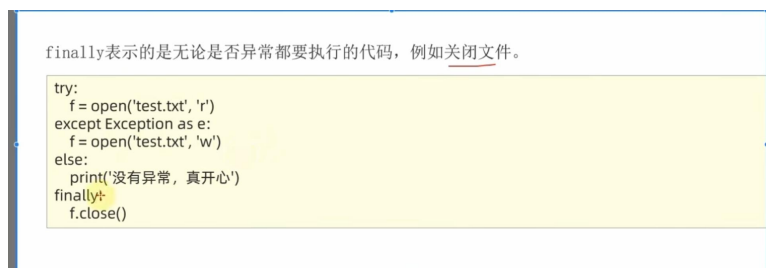
- ① 如果尝试执行的代码的异常类型和要捕获的异常类型不一致, 则无法捕获异常。
- ② 一般try下方只放一行尝试执行的代码。



没有出现异常的处理 else



有无异常都要处理 finally



知道异常信息



异常传递

```
20
21 main()
22
```

```
Traceback (most recent call last):
  File "D:\python-learn\09_异常_模块_包\03_异常的传递.py", line 21, in <module>
    main()
  File "D:\python-learn\09_异常_模块_包\03_异常的传递.py", line 19, in main
    func2()
  File "D:\python-learn\09_异常_模块_包\03_异常的传递.py", line 14, in func2
    func1()
  File "D:\python-learn\09_异常_模块_包\03_异常的传递.py", line 8, in func1
    num = 1 / 0 # 肯定有异常，除以0的异常
```

模块

import 模块名

基本语法:

```
import 模块名
import 模块名1, 模块名2
```

模块名.功能名()

案例: 导入time模块

```
# 导入时间模块
import time

print("开始")
# 让程序睡眠1秒(阻塞)
time.sleep(1)
print("结束")
```

```
import time
print("yy")
time.sleep(5)
print("jj")
```

```
import time as t
print("uu")
t.sleep(3)
print(",,,")
```

1. 什么是模块？

模块就是一个Python代码文件，内含类、函数、变量等，我们可以导入进行使用。

2. 如何导入模块

[from 模块名] import [模块 | 类 | 变量 | 函数 | *] [as 别名]

3. 注意事项：

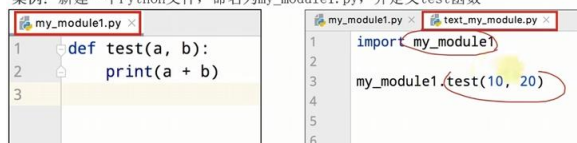
- from可以省略，直接import即可
- as别名可以省略
- 通过"." 来确定层级关系

模块的导入一般写在代码文件的开头位置
去完成组合的导入工作

制作自定义模块

Python中已经帮我们实现了很多的模块。不过有时候我们需要一些个性化的模块，这里就可以通过自定义模块实现，也就是自己制作一个模块

案例：新建一个Python文件，命名为my_module1.py，并定义test函数



The image shows two code editor windows. The left window, titled 'my_module1.py', contains the following code:

```
1 def test(a, b):
2     print(a + b)
3
```

The right window, titled 'text_my_module.py', contains the following code:

```
1 import my_module1
2
3 my_module1.test(10, 20)
4
5
6
```

注意：

同名后面会把前面替换掉

```
# 模块1代码
def my_test(a, b):
    print(a + b)

# 模块2代码
def my_test(a, b):
    print(a - b)

# 导入模块和调用功能代码
from my_module1 import my_test
from my_module2 import my_test

# my_test函数是模块2中的函数
my_test(1, 1)
```


1. 如何自定义模块并导入？

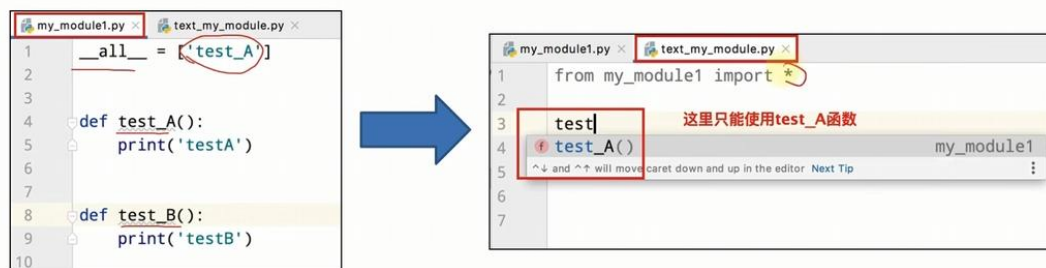
在Python代码文件中正常写代码即可，通过import、from关键字和导入Python内置模块一样导入即可使用。

2. __main__变量的功能是？

if __main__ == "__main__" 表示，只有当程序是直接执行的才会进入if内部，如果是被导入的，则if无法进入

__all__

如果一个模块文件中有`__all__`变量，当使用`from xxx import`导入时，只能导入这个列表中的元素



```
# mymodule.py file
1 个用法
def generate_full_name(firstname, lastname):
    print(firstname + ' ' + lastname)

1 个用法
def test(f,y):
    print(f+y)
__all__=["test"]
if __name__=="__main__":
    generate_full_name( firstname: "3", lastname: "2")
```

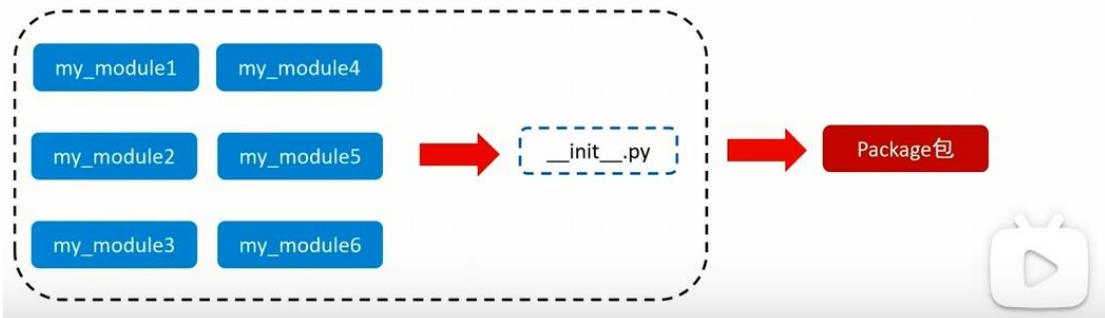
Python 的包

包含一堆模块和一个__init__.py 文件

什么是Python包

从物理上看，包就是一个文件夹，在该文件夹下包含了一个 `__init__.py` 文件，该文件夹可用于包含多个模块文件

从逻辑上看，包的本质依然是模块



导入包

方式一：

```
import 包名.模块名
+
包名.模块名.目标
```

```
my_module1.py × my_module2.py × text_my_module.py ×
1 import my_package.my_module1
2 import my_package.my_module2
3
4 # 包中的my_module1模块的info_print1()方法
5 my_package.my_module1.info_print1()
6 # 包中的my_module1模块的info_print2()方法
7 my_package.my_module2.info_print2()
8
Run: text_my_module ×
```

```
import my_package.my_module
import my_package.my_module2
my_package.my_module.generate_full_name( firstname: "12" lastname: "212")
my_package.my_module2.test( f: 1, y: 3)
```

```
from my_package import my_module
my_module.generate_full_name( firstname: "12" lastname: "212")
from my_package.my_module2 import test
test( f: 1, y: 2)
```

导入包

方式二:

注意: 必须在`__init__.py`文件中添加`__all__ = []`, 控制允许导入的模块列表

from 包名 import *
模块名.目标

```
my_module1.py × my_module2.py × text_my_module.py × __init__.py ×
1 # 包中的__all__和模块中的__all__一样有着控制的功能
2 __all__ = ["my_module2"]
3 |
```

```
my_module1.py × my_module2.py × text_my_module.py × __init__.py ×
1 from my_package import *
2
3 # 包中的my_module1模块的info_print1()方法
4 my_module1.info_print1()
5 # 包中的my_module1模块的info_print2()方法
6 my_module2.info_print2()
```

my_module1报红证明不可

注意:
__all__针对的是'from ...
import xxx'这种方式无

它是不是可以控制我们

```
1 __all__=["my_module2"]
```

```
2 from my_package import my_module
3 my_module.generate_full_name( firstname: "12" , lastname: "212")
4 from my_package import *
5 my_module2.test( f: 2 , y: 3)
6 |
```

1. 什么是Python的包?

包就是一个文件夹, 里面可以存放许多Python的模块(代码文件), 通过包, 在逻辑上将一批模块归为一类, 方便使用。

2. __init__.py文件的作用?

创建包会默认自动创建的文件, 通过这个文件来表示一个文件夹是Python的包, 而非普通的文件夹。

3. __all__变量的作用?

同模块中学习到的是一个作用, 控制 import * 能够导入的内容

常用的包

我们知道，包可以包含一堆的Python模块，而每个模块又内含许多的功能。

所以，我们可以认为：一个包，就是一堆同类型功能的集合体。

在Python程序的生态中，有许多非常多的第三方包（非Python官方），可以极大的帮助我们提高开发效率，如：

- 科学计算中常用的：numpy包
- 数据分析中常用的：pandas包
- 大数据计算中常用的：pyspark、apache-flink包
- 图形可视化常用的：matplotlib、pycharts
- 人工智能常用的：tensorflow

总结

1. 什么是第三方包？有什么作用？

第三方包就是非Python官方内置的包，可以安装它们扩展功能，提高开发效率。

2. 如何安装？

- 在命令提示符内：
 - `pip install 包名称`
 - `pip install -i https://pypi.tuna.tsinghua.edu.cn/simple 包名称`
- 在PyCharm中安装

包的练习

创建一个自定义包，名称为：my_utils（我的工具）

在包内提供2个模块

- str_util.py（字符串相关工具，内含：）
 - 函数：str_reverse(s)，接受传入字符串，将字符串反转返回
 - 函数：substr(s, x, y)，按照下标x和y，对字符串进行切片
- file_util.py（文件处理相关工具，内含：）
 - 函数：print_file_info(file_name)，接收传入文件的路径，打印文件的全部内容，如文件不存在则捕获异常，输出提示信息，通过finally关闭文件对象
 - 函数：append_to_file(file_name, data)，接收文件路径以及传入数据，将数据追加写入到文件中

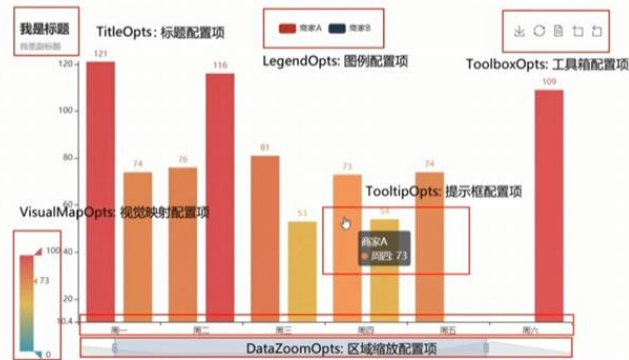
```

1  #字符处理
   1个用法
2  def str_reverse(s):
3      """
4      完成反转
5      :param s: 被反转的字符串
6      :return: 反转后字符串
7      """
8      return s[::-1]
9
10
11 1个用法
12 def substr(s, x, y):
13     """
14     按照给定功能切片字符串
15     :param s: 被切开的字符串
16     :param x: 切片起始下标
17     :param y: 切片结束下标
18     :return:
19     """
20     return s[x:y]
21
22 if __name__ == '__main__':
23     print(str_reverse("call of duty"))
24     print(substr(s: "cod21" x: 1 y: 3))

```

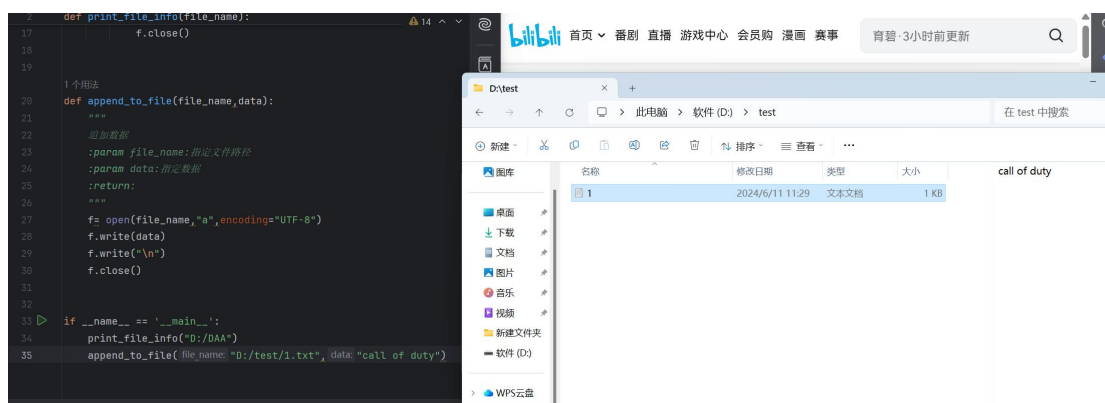
set_global_opts方法

- 这里全局配置选项可以通过set_global_opts方法来进行配置，相应的选项和选项的功能如下：



- 系列配置项，我们在后
- 如下那么假设我们有一个基础的图表

```
def print_file_info(file_name):  
    """  
    将路径文件输出到控制台  
    :param file_name: 读取的文件  
    :return:  
    """  
    f=None  
    try:  
        f=open(file_name,"r",encoding="UTF-8")  
        content=f.read()  
        print(content)  
    except Exception as e:  
        print("wor")  
    finally:  
        if f:  
            f.close()  
  
1个用法  
def append_to_file(file_name,data):  
    """  
    追加数据  
    :param file_name: 指定文件路径  
    :param data: 指定数据  
    :return:  
    """  
    f= open(file_name,"a",encoding="UTF-8")  
    f.write(data)  
    f.write("\n")  
    f.close()  
  
> if __name__ == '__main__':  
    print_file_info("D:/DAA")  
    append_to_file( file_name: "D:/test/1.txt", data: "call of duty")
```




```
test.py x str_util.py file_util.py module.py
1 import my_package.str_util
2 from my_package import file_util
3 print(my_package.str_util.str_reverse("asdfg"))
4 print(my_package.str_util.substr(s: "asdfg", x: 2, y: 4))
5 file_util.append_to_file(file_name: "D:/test/1.txt", data: "rtx4070")
6 file_util.print_file_info("D:/test/1.txt")
7
```

pythonProject

my_package

render.html

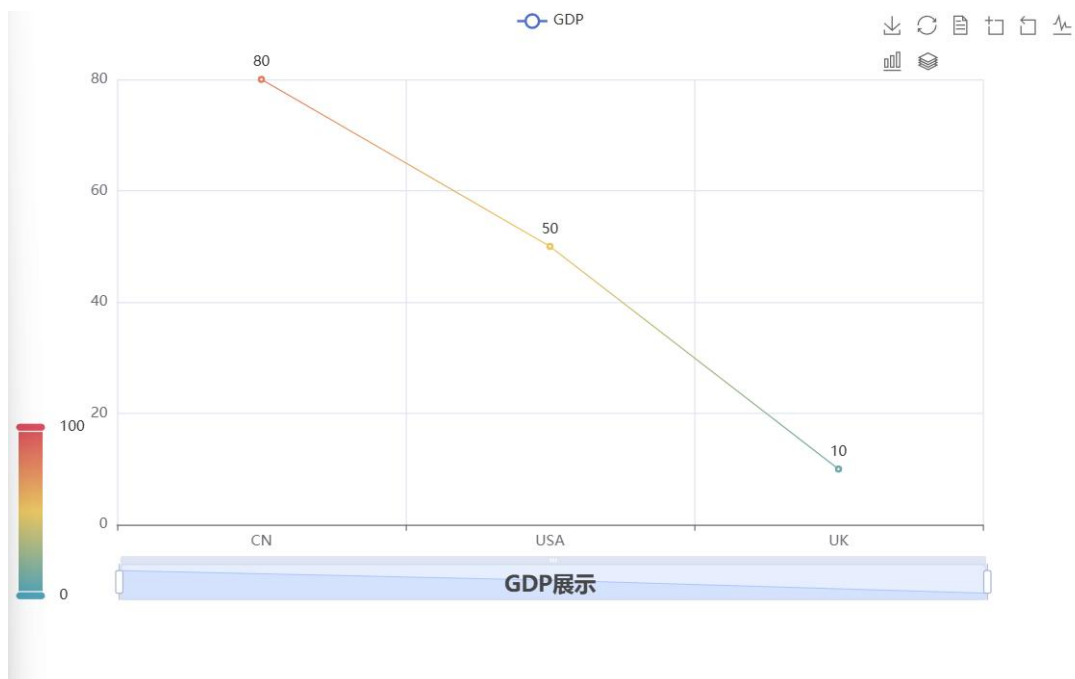
test.py

test

D:\python f\pythonProject\venv\Scripts\python.exe "D:\python f\pythonProject\test.py"

进程已结束，退出代码为 0

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Awesome-pyecharts</title>
<script type="text/javascript" src="https://assets.pyecharts.org/1.2.1/pyecharts.min.js"></script>
</head>
<body>
<div id="5340a23023794193924dc7353d5520b" class="chart-container" style="width: 100%; height: 400px;>
<script>
var chart_5340a23023794193924dc7353d5520b = echarts.init(
document.getElementById("5340a23023794193924dc7353d5520b"), 'white', {
  animation: true,
  animationThreshold: 2000,
  animationDuration: 1000,
  animationEasing: "cubicOut",
  animationDelay: 0,
  animationDurationUpdate: 300,
  animationEasingUpdate: "cubicOut",
  animationDelayUpdate: 0,
  aria: {
    enabled: false
  },
  "color": [
    "#5470c6",
    "#91cc75"
  ]
});
chart_5340a23023794193924dc7353d5520b.setOption({
  title: {
    text: "GDP展示"
  },
  tooltip: {
    trigger: "item"
  },
  legend: {
    type: "none"
  },
  series: [
    {
      name: "GDP",
      type: "bar",
      data: [
        {country: "CN", gdp: 80},
        {country: "USA", gdp: 50},
        {country: "UK", gdp: 10}
      ]
    }
  ]
});
</script>
</body>
</html>
```



```

f_us = open("D:/美国.txt", "r", encoding="UTF-8")
us_data = f_us.read() # 美国的全部内容
# 去掉不合JSON规范的开头
us_data = us_data.replace("jsonp_1629344292311_69436(", "")
# 去掉不合JSON规范的结尾
us_data = us_data[:-2]
# JSON转Python字典
us_dict = json.loads(us_data)
print(type(us_dict))
print(us_dict)

```

