1. 今日内容大纲

第一次班委会总结:

- 微调座位.
- 开展运维兴趣班.
- 培养独立思考问题的能力.
  - 遇到难题 选做题:先思考20分钟左右.关键节点问问思路.
  -

1. 自定义模块
2. 模块是什么?
3. 为什么要有模块?
   - 什么是脚本?
4. 模块的分类
5. import的使用
   - 第一次导入模块执行三件事
   - 被导入模块有独立的名称空间
   - 为模块起别名
   - 导入多个模块
6. from ... import ...
   - from ... import ...的使用
   - from ... import ... 与import对比
   - 一行导入多个
   - from ... import *
   - 模块循环导入的问题
   - py文件的两种功能
   - 模块的搜索路径
7. json pickle 模块
8. hashlib模块

2. 具体内容

1. 自定义模块
2. 模块是什么?

   抖音: 20万行代码全部放在一个py文件中?

   为什么不行?

   ```
   1. 代码太多,读取代码耗时太长.
   ```

   2. 代码不容易维护.

   所以我们怎么样?

一个py文件拆分100文件,100个py文件又有相似相同的功能.冗余. 此时你要将100个py文件中相似相同的函数提取出来, input 功能,print()功能, time.time() os.path.....放在一个文件,当你想用这个功能拿来即用.类似于这个py文件: 常用的相似的功能集合.模块.

模块就是一个py文件常用的相似的功能集合.

3. 为什么要有模块?
   - 拿来主义,提高开发效率.
   - 便于管理维护.
   - 什么是脚本?
     - 脚本就是py文件.长期保存代码的文件.
4. 模块的分类
   1. 内置模块200种左右.Python解释器自带的模块,time os sys hashlib等等.
   2. 第三方模块6000种.一些大牛大神写的,非常好用的.

      pip install 需要这个指令安装的模块,Beautiful_soup,request,Django,flask 等等.
   3. 自定义模块,自己写的一个py文件.
5. import的使用

   import 模块 先要怎么样?

   import tbjx 执行一次tbjx这个模块里面的所有代码,

   第一次引用tbjx这个模块,会将这个模块里面的所有代码加载到内存,只要你的程序没有结束,接下来你在引用多少次,它会先从内存中寻找有没有此模块,如果已经加载到内存,就不在重复加载.

   - 第一次导入模块执行三件事 `***`
     - 1. 在内存中创建一个以tbjx命名的名称空间.
       2. 执行此名称空间所有的可执行代码(将tbjx.py文件中所有的变量与值的对应关系加载到这个名称空间).
       3. 通过tbjx. 的方式引用模块里面的代码.

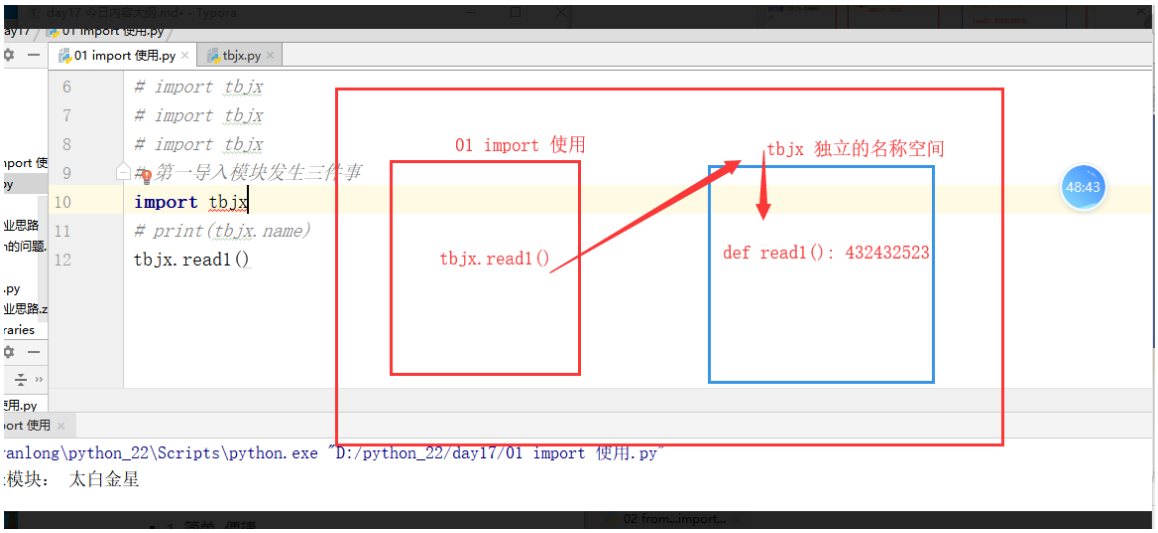
```
import tbjx
print(tbjx.name)
tbjx.read1()
```

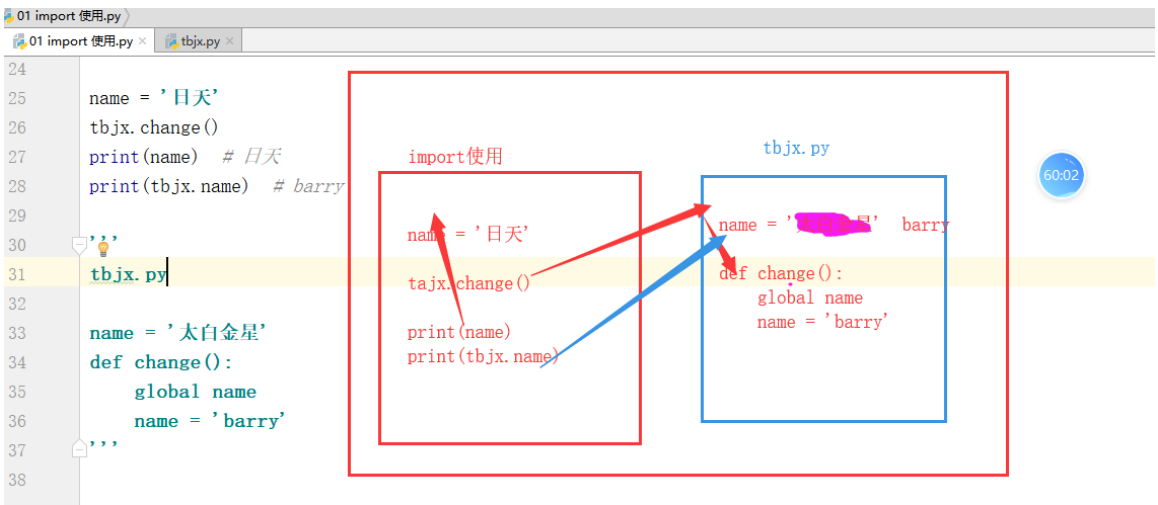
- 被导入模块有独立的名称空间 ***

```
import tbjx
# name = 'alex'
# print(name)
# print(tbjx.name)

#
# def read1():
#     print(666)
# tbjx.read1()
#

# name = '日天'
# tbjx.change()
# print(name)  # 日天
# print(tbjx.name)  # barry
```





- 为模块起别名 **
  - 1 简单,便捷.

```
# import hfjksdahdsafkd as sm
# print(sm.name)
```

- 2,有利于代码的简化.

```
# 原始写法
# result = input('请输入')
# if result == 'mysql':
#     import mysql1
#     mysql1.mysql()
# elif result == 'oracle':
#     import oracle1
#     oracle1.oracle()
# list.index()
# str.index()
# tuple.index()
# 起别名
# result = input('请输入')
# if result == 'mysql':
#     import mysql1 as sm
# elif result == 'oracle':
#     import oracle1 as sm
# ''' 后面还有很多'''
# sm.db()  # 统一接口,归一化思想
```

- 导入多个模块

```
import time, os, sys  # 这样写不好
# 应该向以下这种写法:
import time
import os
import sys
```

6. from ... import ...

- from ... import ...的使用

```
# from tbjx import name
# from tbjx import read1
# from tbjx import read2
# print(name)
# print(globals())
# read1()
```

- from ... import ... 与import对比 ***

1. from.. import 用起来更方便

```
from tbjx import name
print(name)
```

2. from...import 容易与本文件的名字产生冲突.

```
# 1, 容易产生冲突,后者将前者覆盖
# name = 'alex'
# from tbjx import name
# print(name)
```

3. 当前位置直接使用read1和read2,执行时，仍然以tbjx.py文件全局名称空间 `***`

```
# from tbjx import read1
#
# def read1():
#     print(666)

# name = '大壮'
# read1()
# print(globals())


from tbjx import change
name = 'Alex'
print(name)  # 'Alex'
change()  # 'barry'
from tbjx import name
print(name)
```

- 一行导入多个

```
from tbjx import name,read1,read2  # 这样不好
from tbjx import name
from tbjx import read1
```

- from ... import *
- 模块循环导入的问题

- py文件的两种功能

py文件的两个功能：

1. 自己用 脚本
2. 被别人引用 模块使用

```
# print('from the tbjx.py')
__all__ = ['name', 'read1']  # 配合*使用

name = '太白金星'

def read1():
    print('tbjx模块: ', name)

def read2():
    print('tbjx模块')
    read1()

def change():
    global name
    name = 'barry'
    print(name)



# print(__name__)
# 当tbjx.py做脚本: __name__ == __main__  返回True
# 当tbjx.py做模块被别人引用时: __name__ == tbjx

# __name__ 根据文件的扮演的角色(脚本,模块)不同而得到不同的结果
#1, 模块需要调试时,加上 if __name__ == '__main__':
# import time

# change()  # 测试代码
# if __name__ == '__main__':
#     change()

# 2, 作为项目的启动文件需要用.
```

- 模块的搜索路径 `***`

```
# import sm
import abc
# python 解释器会自动将一些内置内容(内置函数,内置模块等等)加载到内存中
import sys
# print(sys.modules)  # 内置内容(内置函数,内置模块等等)
import time

# print(sys.path)
#['D:\\python_22\\day17', 'C:\\Python\\Python36\\python36.zip',
'C:\\Python\\Python36\\DLLs', 'C:\\Python\\Python36\\lib',
'C:\\Python\\Python36', 'C:\\Python\\Python36\\lib\\site-packages']
```

```
# 'D:\\python_22\\day17' 路径是当前执行文件的相对路径
# import tbjx

# 我就想找到dz 内存没有,内置中,这两个你左右不了,sys.path你可以操作.
import sys
sys.path.append(r'D:\python_22\day16')
# sys.path 会自动将你的 当前目录的路径加载到列表中.
import dz

# 如果你想要引用你自定义的模块:
# 要不你就将这个模块放到当前目录下面,要不你就手动添加到sys.path
```

- import sm
    1. 它会先从内存中寻找有没有已经存在的以sm命名的名称空间.
    2. 它会从内置的模块中找. time,sys,os,等等.
    3. 他从sys.path中寻找.

7. json pickle 模块

序列化模块: 将一种数据结构(list,tuple,dict ....)转化成特殊的序列.

为什么存在序列化?

数据 ----> bytes

只有字符串类型和bytes可以互换.

dict,list..... -------> str <---------> bytes

数据存储在文件中,str(bytes类型)形式存储,比如字典.

数据通过网络传输(bytes类型),str 不能还原回去.

特殊的字符串:序列化.

序列化模块:

json模块 : （**重点**）

1. 不同语言都遵循的一种数据转化格式，即不同语言都使用的特殊字符串。（比如Python的一个列表[1, 2, 3]利用json转化成特殊的字符串，然后在编码成bytes发送给php的开发者，php的开发者就可以解码成特殊的字符串，然后在反解成原数组(列表): [1, 2, 3])
2. json序列化只支持部分Python数据结构: dict,list, tuple,str,int, float,True,False,None

```
l1 = [i for i in range(10000)]
# l1 --->bytes
# b1 = l1.encode('utf-8')  # 不能直接转换
# l1转化成字符串在转化成bytes
s1 = str(l1)
b1 = s1.encode('utf-8')
# print(b1)

# 岑哥接收了b1
s2 = b1.decode('utf-8')
# print(s2,type(s2))
```

```python
# str 我们学过的str

# dic = {'username': '太白', 'password': 123,'status': True}
import json
#dumps loads 主要用于网络传输,但是也可以读写文件
# 特殊的字符串
# st = json.dumps(dic,ensure_ascii=False)
# print(st,type(st))
# # 反转回去
# dic1 = json.loads(st)
# print(dic1,type(dic1))

# 写入文件
# l1 = [1, 2, 3, {'name': 'alex'}]
# 转化成特殊的字符串写入文件
# with open('json文件',encoding='utf-8',mode='w') as f1:
#     st = json.dumps(l1)
#     f1.write(st)

# 读取出来还原回去
# with open('json文件',encoding='utf-8') as f2:
#     st = f2.read()
#     l1 = json.loads(st)
#     print(l1,type(l1))
# 特殊的参数
l1 = [1, 2, 3, {'name': 'alex'}]

# dump load 只能写入文件,只能写入一个数据结构
# with open('json文件1',encoding='utf-8',mode='w') as f1:
#     json.dump(l1,f1)

# 读取数据
# with open('json文件1',encoding='utf-8') as f2:
#     l1 = json.load(f2)
#     print(l1,type(l1))


# 一次写入文件多个数据怎么做?

# 错误示例:
# dic1 = {'username': 'alex'}
# dic2 = {'username': '太白'}
# dic3 = {'username': '大壮'}
# with open('json文件1',encoding='utf-8',mode='w') as f1:
#     json.dump(dic1,f1)
#     json.dump(dic2,f1)
#     json.dump(dic3,f1)


# 读取数据
# with open('json文件1',encoding='utf-8') as f1:
#     print(json.load(f1))
```

```
    # print(json.load(f1))
    # print(json.load(f1))


# 正确写法:
dic1 = {'username': 'alex'}
dic2 = {'username': '太白'}
dic3 = {'username': '大壮'}
# with open('json文件1',encoding='utf-8',mode='w') as f1:
#     f1.write(json.dumps(dic1) + '\n')
#     f1.write(json.dumps(dic2) + '\n')
#     f1.write(json.dumps(dic3) + '\n')

# with open('json文件1',encoding='utf-8') as f1:
#     for i in f1:
#         print(json.loads(i))
```

pickle模块:

1. 只能是Python语言遵循的一种数据转化格式，只能在python语言中使用。
2. 支持Python所有的数据类型包括实例化对象。

```
l1 = [1, 2, 3, {'name': 'alex'}]

# dumps loads 只能用于网络传输
# import pickle
# st = pickle.dumps(l1)
# print(st)  # bytes
#
# l2 = pickle.loads(st)
# print(l2,type(l2))

# dump load 直接写入文件
# import pickle
# dic1 = {'name':'oldboy1'}
# dic2 = {'name':'oldboy2'}
# dic3 = {'name':'oldboy3'}
#
# f = open('pick多数据',mode='wb')
# pickle.dump(dic1,f)
# pickle.dump(dic2,f)
# pickle.dump(dic3,f)
# f.close()
# import pickle
# f = open('pick多数据',mode='rb')
# print(pickle.load(f))
# print(pickle.load(f))
# print(pickle.load(f))
```

```
# f.close()

import pickle
def func():
    print('in func')

# f = open('pick对象',mode='wb')
# pickle.dump(func,f)
# f.close()

# f = open('pick对象', mode='rb')
# ret = pickle.load(f)
# print(ret)
# ret()
```

8. hashlib模块

   包含很多的加密算法. MD5, sha1 sha256 sha512......

   用途:

   1. 密码加密.不能以明文的形式存储密码.密文的形式.
   2. 文件的校验.

   用法:

   1. 将bytes类型字节 转化成 固定长度的16进制数字组成的字符串.
   2. 不同的bytes利用相同的算法(MD5)转化成的结果一定不同.
   3. 相同的bytes利用相同的算法(MD5)转化成的结果一定相同.
   4. hashlib算法不可逆(MD5中国王晓云破解了).

```
# import hashlib
# hashlib.md5()

# md5
# s1 = 'kfdslfjasdlfgjsdlgkhsdafkshdafjksdfsdkfhjsdafj老fhdskafhsdkjfdsa男孩教育'
# import hashlib
# ret = hashlib.md5()
# ret.update(s1.encode('utf-8'))
# print(ret.hexdigest(),)

# 相关练习
import hashlib
# def MD5(pwd):
#     ret = hashlib.md5()
#     ret.update(pwd.encode('utf-8'))
#     return ret.hexdigest()
#
#
# def register():
#     username = input('请输入用户名:').strip()
#     password = input('请输入密码:').strip()
#     password_md5 = MD5(password)
#     with open('register',encoding='utf-8',mode='a') as f1:
```

```
#         f1.write(f'\n{username}|{password_md5}')
#
# register()
#
# def login():
#     username = input('请输入用户名:').strip()
#     password = input('请输入密码:').strip()
#     password_md5 = MD5(password)
#

# 普通加密

import hashlib
# s1 = 'kfdslfjasdlfgjsdlgkhsdafkshdafjksdfsdkfhjsdafj老fhdskafhsdkjfdsa男孩教育'
# s2 = 'kfdslfjasdlfgjsdlgkhsdafkshdafjksdfsdkfhjsdafj老fhdskafhsdkjfdsa男孩教育'
# ret = hashlib.md5()
# ret.update(s2.encode('utf-8'))
# print(ret.hexdigest())  # 18f127c24462dd59287798ea5c0c0c2f
18f127c24462dd59287798ea5c0c0c2f
# 123456:  18f127c24462dd59258898ea5c0c0c2f
# OOOOOO:  18f127c24462dd59258898we5c0c0c2f

# s2 = '19890425'
# ret = hashlib.md5()
# ret.update(s2.encode('utf-8'))
# print(ret.hexdigest())  # 6e942d04cf7ceeeba09e3f2c7c03dc44

# 加盐

# s2 = '19890425'
# ret = hashlib.md5('太白金星'.encode('utf-8'))
# ret.update(s2.encode('utf-8'))
# print(ret.hexdigest())  # 84c31bbb6f6f494fb12beeb7de4777e1


# 动态的盐
# s2 = '19890425'
# ret = hashlib.md5('太白金星'[::2].encode('utf-8'))
# ret.update(s2.encode('utf-8'))
# print(ret.hexdigest())  # 84c31bbb6f6f494fb12beeb7de4777e1


# sha系列  金融类,安全类.用这个级别.
# 随着sha系列数字越高,加密越复杂,越不易破解,但是耗时越长.
# s2 = '198fdsl;fdsklgfjsdlgdsjlfkjsdalfksjdal90425'
# ret = hashlib.sha3_512()
# ret.update(s2.encode('utf-8'))
# print(ret.hexdigest())  #
4d623c6701995c989f400f7e7eef0c4fd4ff15194751f5cb7fb812c7d42a7406ca0349ea3447d245ca29b48a941
e2f2f66579fb090babb73eb2b446391a8e102
```

```
# 文件的校验

# linux中一切皆文件: 文本文件,非文本文件,音频,视频,图片....
# 无论你下载的视频,还是软件(国外的软件),往往都会有一个md5值


# 6217ce726fc8ccd48ec76e9f92d15feecd20422c30367c6dc8c222ab352a3ec6  *pycharm-professional-
2019.1.2.exe

# s1 = '我叫太白金星 今年18岁'
# ret = hashlib.sha256()
# ret.update(s1.encode('utf-8'))
# print(ret.hexdigest())  #
54fab159ad8f0bfc5df726a70332f111c2c54d31849fb1e4dc1fcc176e9e4cdc
#
# ret = hashlib.sha256()
# ret.update('我叫'.encode('utf-8'))
# ret.update('太白金星'.encode('utf-8'))
# ret.update(' 今年'.encode('utf-8'))
# ret.update('18岁'.encode('utf-8'))
# print(ret.hexdigest()) # 54fab159ad8f0bfc5df726a70332f111c2c54d31849fb1e4dc1fcc176e9e4cdc

# low版校验:
def file_md5(path):
    ret = hashlib.sha256()
    with open(path,mode='rb') as f1:
        b1 = f1.read()
        # print(b1)
        ret.update(b1)
    return ret.hexdigest()
result = file_md5('pycharm-professional-2019.1.2.exe')
print(result)  # 6217ce726fc8ccd48ec76e9f92d15feecd20422c30367c6dc8c222ab352a3ec6

# 高大上版
```

1. 今日总结

1. import 三件事, import的名字如果调用? 模块名.的方式调用.
2. from ... import ... 容易产生冲突,独立的空间.
3. `__name__`问题
4. 模块的搜索路径

   内存 内置 sys.path
5. 序列化模块:
   1. json最最常用(两对四个方法就行) 一定要掌握
   2. pickle(两对四个方法就行) 尽量掌握
6. hashlib

1. 密码的加密 ,文件的校验

2. 预习内容

规范化目录结构