字符串操作：

转换成大写：

```
my_str = "j2hdksil mk"
res = my_str.upper()
```

转换成小写：

```
res = my_str.lower()
```

首字母大写

```
res3 = my_str.capitalize()
```

将每个单词的首字母大写

```
res4 = my_str.title()
```

字符串的切分 2对应参数位置 表示的是切分几次

```
res5 = my_str.split(" ",2)
```

去掉左侧指定字符、右侧

```
res9 = my_str.lstrip("ss")
print(res9)

### 右侧
res10 = my_str.rstrip(" ")

### 同时去掉两侧的字符
my_str = ''
res11 = my_str.strip('s')
```

字符串连接

```
res12 = "+".join(my_str)
```

左右对齐 与 补0对齐

```
# 左对齐
my_str = "we"
res11 = my_str.ljust(4, '-')
print(res11)
```

```
5
6  # 右对齐
7  res12 = my_str.rjust(4, '+')
8  print(res12)
9
10  # zfill 将字符串右对齐 然后位数不够用0补
11  注意当指定位数小于字符串长度 使用的是字符串长度本身
12  my_str = "aq1"
13  res16 = my_str.zfill(2)
```

居中

如果字符串长度小于指定的长度，默认是用空格然后也可以指定填充的字符

```
1  res13 = my_str.center(5, "ww")
2  print(res13)
```

字符串的替换replace

replace('你要替换的字符串' ， '用什么替换' ， 替换多少个')

```
1  res14 = my_str.replace("h",'8')
2  print(res14)
3  res15 = my_str.replace('s', '6', 2)
4  #解释：‘s’ 是我要替换的 用‘6’来替换， 2说的是替换的个数
```

大小写互相转换

```
1  my_str = "AcjjU"
2  res = my_str.swapcase()
```

切分

按照第一个匹配的字符去切分 保留指定切分的切分字符， 变成元组,元组第一项是切分字符左侧的内容 第二项是你指定切分字符 第三项是切分字符右侧的内容

```
1  res18 = my_str.partition("c")
```

统计字符出现的个数

```
1  res20 = my_str.count("1",起始值，结束值)
```

格式化输出

```
age = 18
money = 100000000
print("我的年纪是%d" % age)
# 多个值的情况 “字符串%d呵呵呵%d” % (age, money)
print("我的年纪是%d,我的存款%d津巴布韦币" % (money, money))
```

%后的变量是依次赋值给你的占位符

```
%d 整数
%f float类型 默认输出小数部分是6位 如果想保留自己制定长度 那么使用%.位数
%s 字符串
```

使用format

```

name = "张三"
age = 30
weight = 80.2
my_str1 = "我叫{}, 我今年{}岁,  我体重是{}".format(name, age, weight)

```

dict操作

my_dict = {"name":"lucy","age":20}

```
##clear() 清空字典

##frokeys 生成一个新的字典 第一个参数 是指定你的key序列，第二个参数是字典对应的value
 my_dict1 = {}
 res = my_dict1.fromkeys([1,2,3], "j")

```

items() 将字典转成列表嵌套元组的形式

```
res = my_dict.items()
print(res)
输出为:  dict_items([('name', 'lucy'), ('age', 20)])
```

key(),values()

```
获取字典所有的键
```

```
2 获取字典所有的值
```

pop(key):弹出一个key对应的value值, 原来字典将会删除掉对应的键值对

```
1 res = my_dict.pop("name")
2 print(res)
3 print(my_dict)
```

popitem() 弹出字典的一项 然后原来的字典 将会移除掉对应键值对

setdefault (key, value)

在原来字典的基础上 添加一个键值对 如果key 存在将不会设置成功， 如果key不存在 那么才生效,

函数操作的返回结果 如果key存在是 原来这个key对应value 否则返回的是你设置的value

```
1 res = my_dict.setdefault("name", 'lili')
2 print(res, my_dict)
```

update(字典) :在原来字典的基础上追加新的键值对 如果key有相同的那么 更新原来字典key所对应的value

```
1 my_dict2 = {"money":10, "hobby":[1,2,34], 'name':"bob"}
2 my_dict.update(my_dict2)
3 print(my_dict)
```

list操作

count() 计数 来统计列表元素出现的次数

extend（另一个列表） 将另一个list追加到调用列表后边

```
1 a = ["abc", "abcd", "e", "as"]
2 b = [1,2,4]
3 a.extend(b) 将b列表添加到a里
```

index（要找的元素，找的起始值，找的末位值）

返回你要查找元素的下标（返回的是匹配到的第一个元素） 如果元素不在列表里 那么他将会报一个ValueError

```
res = a.index("absssc",2, len(a))
```

insert（要插入的位置，要插入的元素）

当给定的index较大的时候 那么追加到数组的最后

```
a.insert(10002, "9")
```

pop(需要弹出的下标)

返回弹出的元素 在原来的列表删除对应的下标元素

```
a.pop(0)
```

remove（要移除的元素） 不返回 删除的元素（删除的是他匹配的第一个）

如果要移除的元素没在列表里 那么将会报 ValueError异常

```
res = a.remove("abc")
```

reverse（ ）

将列表反转 逆序 没有返回值 而是直接影响原字符串

```
a.reverse()
```

sort（ ）

排序 默认是对数组里的元素的ascll码比较

如果指定了key参数（排序规则），就使用新的规则，

reverse来设置是不是降序排列 默认False是升序

```
指定排序的规则
def str_len(my_str):
 return len(my_str)
排序
res = a.sort(key=str_len, reverse=True)
print(res, a)
```

列表生成式

```
res1 = [i for i in range(0,10,2) if i %2 ==0]
```

字典生成式

```
a = [(1,2),("name",""kobe")]
res = {key:value for key,value in a}
```

可迭代对象

通俗的讲 就是for可以直接遍历的 字符串 list dict set tuple

判断：

isinstance（变量名， 类名）

功能：帮助我们判断变量是不是 这个类名的实例

Iterable

迭代器

列表生成式 能帮我们快速生成一个列表 但是是全都在内存 如果列表长度过大 而是用的数量还少 那将对内存造成严重的浪费

```
python给我们提供了迭代器来优化此问题：不浪费内存，当你使用的时候 他才给你计算拿出来
声明：1 将列表生成式的中括号 变成小括号
 2 函数与yield关键字组合 形成一个迭代器


yield与return区别：return 结束函数
 而yield只是停止了迭代器（函数）的调用
 都可以返回值
 访问：使用内置next(迭代器对象)函数
  当我们的迭代器对象为空的时候 报错
```

元组

当我们直接使用括号表示一个元组的时候 如果只有一个元素 为防止歧义 要在元素后加一个逗号

操作

元组不可以修改

但可以使用+或*

可以通过下标访问

count（元素）：统计元素在元组里出现的次数

index（元素）：查找第一个匹配到的元素下标 如果不在元组 会报错

变量

局部变量 全局变量

global 关键字

在函数内用于global修饰的变量是全局变量

```
a = 10
def fun1（）：
 global a
a = 6
fun1()
print(a) 6
```

```
闭包中 如果内部函数想去修改外部函数的变量，
那我们就需要使用nonlocal来修饰我们想要修改的变量
def fun3():
 b = 10
 def inner():
 nonlocal b
 b = 5
 inner()
 print(b)
 return inner
```

偏导函数

作用：帮我们'冻结'函数的参数，然后得到新的函数 在调用得到的新的函数时 之需要写剩余的参数就可以

```
print_new = partial(print, end="!", sep="+")
print_new("sjs", "jj")
```

装饰器

装饰器带参数（选修）

```
def out(装饰器需要的参数):
 def dec_two(fun):
 def inner(*args):
 if 装饰器参数 写一些逻辑:
 fun(*args)
 else:
 print("权限不够")
 return inner
 return dec_two
```

模块导入

```
from xxx import 成员 as 别名

当from xxx import * 的时候 如果xxx里的一些成员不想被外部使用 那么可以使用
__all__ = ['放你想暴露的成员']
```

程序入口

我们程序运行的开始位置

```
__name__ 在主文件 打印结果是__main__
 在非主文件（模块文件） 打印的时候是模块的名字

if __name__ == "__main__":

```

sorted

排序 不改变原来的值 产生新的列表

普通列表排序

```
a = ["abd",'ks', 'jeje']
# a.sort(key=len, reverse=True)
```

字典

```
dict_data = {"语文":59, "数学": 79, "化学":90, "物理":30, "python": 100}
# if "yu" in dict_data:
res = sorted(dict_data.items(), key=lambda item:item[1])
```

高阶函数

map(func,序列)

```
[func(i) for i in 序列]
功能：将序列的每一项放入到func这个函数里执行
```

reduce(func,序列)

```
功能：func函数 接收两个参数，将两个数结果累计然后继续与下一个元素结果累计的过程
res = reduce(add, [1,2, 3, 4, 5])
相当于：(((1+2)+3)+4)
```

filter过滤

```
filter(func,序列)
func需要一个参数 如果func返回true 保留 否则剔除
```

os模块　sys解析命令行参数

获取当前路径

```
res = os.curdir
print（res）
```

删除文件

```
os.remove
```

获取文件大小

```
file_size = os.path.getsize(path)
```

os的path操作（比较重要）

```
from os import path as p
判断文件或目录是否存在
res = p.exists（r"E:\工作目录\基础阶段\day10\资料\test\a\666.txt"）

#获取当前程序运行的文件名字
current_file_name = basename(__file__)
print(current_file_name)


```

read函数

如果参数不写　那么文件全部读取出来　如果给了数字　那就读取规定个数的字符

注意：如果不想写close　可以使用with open（ ）as 句柄这样形式

```
with open("lal.txt", "r") as f2, open("lal.txt", "rb") as f4:
 print(f2.read())
 print(f4.read(2).decode("gbk"))
```

写操作

write（字符）

```
with open("heheda.txt", "w") as f1:
 f1.write("再带你们一周!!!")

# with open("append.txt", "a") as f2:
# f2.write("需要发票！！！滴滴的也行\nuuuuu")
#
f3 = open("new.txt", "a+")
data = ["新诗\n", "你是我永远都不想修复的bug\n", "爱你就是个死循环\n"]
```

循环读取

解决文件过大　不能一次性读取的问题

```
import os
path = r"E:\工作目录\基础阶段\day10\资料\作业\youbian.txt"
```

```
f1 = open(path,"r",encoding="utf-8")
has_read = 0
#获取文件大小
file_size = os.path.getsize(path)
READ_NUM = 100
while has_read < file_size:
 res = f1.read(READ_NUM)
  if len(res)<= 0:
  pass
  else:
  print(res)
  has_read += READ_NUM
```

解归档

把我们程序里面的数据结构 完整保存到我们文件里

以及读取

使用pickle包

```
import pickle

a = {"hehe":"88","age":12}
#归档
my_file = open("my_pickle.txt","wb")
pickle.dumo(a,my_file)

#解档
my_file = open("my_pickle.txt","rb")
 res = pickle.load(my_file)
 print(res,type(res))
 my_file.close()
```

类的实例化

在程序里定义变量 保存到了栈 但是类的数据是在堆中 在栈里保存了你实例化对象的数据地址

```
1
```

成员函数私有化 @property

把方法变成属性 可以直接用.操作

继承

```
from my_person import Humen
class 子类名(父类名):
 def __init__(self,name,age,sex,school):
 #调用父级初始化函数
 super().__init__(name, age, sex)
 """
 super() 表示父类
 """
 # 调用父类初始化函数的另一种方案
  父类名字.__init__(self, name, age, sex)
 # 自己独有的属性 我们就照常去写
  self.school = school
```

注意 父类私有化的东西 不能在子类里修改

枚举类型

——列举有限的集合

```
from enum import Enum, IntEnum, unique

@unique
class Week(Enum):
 MON = "周一"
 TUS = "周二"
 WEN = "周三"
 THU = "周四"
 FRI = "周五"
  SAT = "周六"
```

```
    SUN = "周天"

# 访问
res = Week.MON.value
print(res)
```

异常处理

```
try:
    是我们想要捕获异常代码段（有可能出现问题的地方）
except ValueError as 变量名:
    如果程序出现异常下边的代码将会被执行 如果程序没出错 那么下边的代码 不会被执行
    print("错误", 变量名)
变量名里 包含了我们的错误信息
```

except可以写多个并且 只要有一个被捕获的类型满足条件 那么其他的except将不会被执行 示例如下

```
try:
    # int("9o")
    print(a)
except ValueError as e: 在python2里as可以使用,来替代 python3不可以
    print("错误", e)
except NameError as e:
    print("变量不存在", e)
except IndexError as e:
    print("下标错误")
else:
    当程序不出错的时候 会进入下边的代码
finally:
    不管程序是否正常 最后都需要执行此处的代码
```

自己手动抛出异常

raise 异常类型的类

```
raise Exception（"字符串位数不正确"）
```

自定义异常

```
class LengError(Exception):

 def __init__(self, msg):
 # super().__init__(msg)
 self.msg = msg

 def __str__(self):
 return "[LengError]:详情:{}".format(self.msg)



raise LengError("长度不够")
或者
a = "12"
try:
 if len(a) != 3:
 raise LengError("呵呵")
except LengError as e:
 print(e,"___")
```