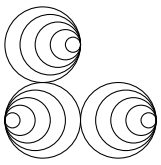
Python 基础知识

Tong Zhang Cls

笔记整理与汇总

版权声明



目录

第一章　数据基本类型

1.1　数字类型

- `Type()` 查询对象类型
- **int** 整形，任意大小的整数
- **Float** 浮点数，小数。

  可写为 `4.3e-3` 的形式。
- **Bool** 布尔值，有 `True` 和 `False`
- **Complex** 复数，可以用 `complex(a,b)` 表示。

  使用 `n.real` 获取实部，`n.imag` 获取虚部
- **None** 空值，是一个特殊的值。

1.2　运算符

1.2.1　算术运算符

- 取模% 返回除法的余数

  例如 `a=10; b=21; b%a=1`
- 取整除// 向下取接近除数的整数

  例如 `9//2=4; -9//2=-5`

1.2.2　赋值运算符

- `+= c+=a` 即 `c=c+a`
- `-= c-=a` 即 `c=c-a`
- `%= c%=a` 即 `c=c%a`

1.2.3 表达式-类型转换

- Int() 整形，舍弃小数部分
- Float() 浮点
- Complex() 复数
- Hex() 十六进制，或者输入 0O
- Oct() 八进制，或者输入 0X
- Bin() 二进制，或者输入 0b
- Chr() 转换成整数对应的 ASCII 字符
- Ord() 转换成 ASCII 对应的整数

1.3 常见内置函数

- eval() 自动类型转换
- Abs() 绝对值
- Max(a,b,c) 最大值
- Min(a,b,c,d) 最小值
- Pow(x,y) x**y 运算
- Round(x, n) 返回浮点数 x 的四舍五入值，如给出 n 的值，代表舍入到小数点后的位数。
- Divmod(x,y) 商余，同时输出商和余数，输出形式为 (3,1)

  例如 divmod(10,3) → (3,1)
- len(s) 返回对象长度或项目个数
- str() 任意类型 x 对应的字符串形式

第二章　字符串与列表

2.1　字符串

- 字符串的索引 `s="abcdefg"` 使用 `s[1]` 输出 `b`
- 多行字符串 起止是 3 个单引号 `'''abcdef'''`
- 字符串运算符切片运算符
  - `s[0:5]` 得到包含 `s[0]` 到 `s[4]` 的全部内容
  - `[:N]` 从头开始
  - `[:3]` 到第二个为止
  - `[:]` 整个字符串
  - `[M:N:K]` 根据**步长**对字符串切片
  - `[1:8:2]` 一隔一取字符
  - `[::-1]` **反向**（-1 表示倒着取）
  - `[0:-1]` 从 0 到最后一个前一个

2.2　字符串基本操作

2.2.1　基本函数

- `len(s)` 返回对象长度或项目个数（字符，列表，元组）
- `str(x)` 任意类型 x 对应的字符串形式，转换为字符串
- `oct(x)` 以八进制小写形式输出
- `hex(x)` 输出为十六进制
- `chr(u)` u 为**整数**，返回对应字符的 Unicode 编码
- `ord(x)` 输入**字符 x**，返回其对应的 Unicode 编码

2.2.2　大小写转换

- `upper()` 转换为**大写**
- `lower()` 转换为**小写**

- swapcase() 翻转大小写
- capitalize() 把字符串的**第一个字符**大写
- isupper() 若**都是大写**，则返回 True
- islower() 若都是小写，则返回 True
- title() 所有单词**以大写开始**，其余字母均为小写
- istitle() 若 x 标题化，返回 True，否则返 False

2.2.3 字符类型判断

- isalpha() **只包含字母**，且非空，返回 True
- isalnum() **只含字母和数字**，且非空，返回 True
- isdigit() **只包含数字**字符，且非空，返回 True
- isspace() **只包含空格、制表符和换行**，且非空

2.2.4 字符串基本方法

- str.split(X) 返回**按照 X 对 str 进行切割**列表

  "A,B,C".split(",") → ['A','B','C']

- str.splitlines() 按照**换行符**来切分字符串
- str.count(X) 输出字串 X 在 str 中**出现的次数**
- str.replace(老，新) **所有 old 字串被替换为 new**，替换每一个。此处被视为字串对待

  ```
  "aa bb cc dd".replace(" ",",")
  "aa,bb,cc,dd"
  ```

- str.strip(chars) 从字符串**左右两边同时去掉** chars 列出的字符。当检测到不在 chars 中的字符时即停，忽略中间的 chars。此处为 chars 含有的每一个字符。

  ```
  "aabbccddaabbccdd".strip("adc")
  'bbccddaabb'
  ```

- str.lstrip() 从左侧去掉列出的字符
- str.rstrip() 从右侧去掉列出的字符
- str.join(iter) 将 iter 变量除最后一个元素外**每个元素后增加一个 str**

  ",".join("12345") '1,2,3,4,5'

2.2.5 字符串查找

- string.find(str,beg=0,end=len(string))
  检查 str 是否包含在 **string** 中，如果 **beg** 和 **end** 指定范围，则检查是否包含在指定范围内，如果是则返回开始的索引值，否则返回-1
- string.rfind(str) 从右往左找

2.3 列表及其基本操作

使用方括号 列表名 =[1,2,3.....] 其中甚至可以放任何支持迭代的容器或对象，例如：

```
week1=['a','b'] week2=[x for x in week1]
```

2.3.1 添加元素

- append(元素) **追加**将一个新的元素添加到列表的尾部，**直接修改，没有返回值**
- insert(位置，元素) **插入**用法：insert(0,"a") 在 0 位置添加一个元素，其余后移一位
- extend() **扩展**列表，相当于连接符 +
  例如 week1=[a]; week2=[b];
  week1.extend(week2) 则修改 week1

2.3.2 删除元素

- Remove() **移除**其根据一个**元素的内容**进行删除。若元素不在列表里，则会异常报错。

  为避免错误可提前进行判断，如
  if(item in week): week.remove(item)
  week.remove('星期六') 删除星期六
  从左往右删除，仅删除第一个

- Pop() **弹出**根据**下标**删除单个或多个函数。

  pop 方法有返回值，会返回被删除的数据，**同时更改原始数据**（这是两个操作）
  例如 item=week.pop(2) 其删除元素后，能够返回被删除的元素
  括号里可不带值，删除最后一个元素

2.3.3 列表查询

- count.(x) 统计 x 出现次数
- index.(x,a,b) 返回从开始位 a 开始到结束位 b 结束的 x 的具体位置。有异常错误，需确保存在性

2.3.4 列表复制

- **错误示例** `week1=[] week2=week1`

  若运行上述语句，若修改 week1，则 week2 **同步变化**。数据在内存中仅有一份

- **切片法** `week2=week1[:]` 实现部分数据复制
- `a.copy()` 方便的复制方法。`week2=week1.copy()`

  返回**浅拷贝**：`[a,[ ],c]` 这类列表无法通过 `[:]`、`copy` 进行复制，减少内存占用

- **深拷贝**：需要调用包：`deepcopy`

2.3.5 列表排序

- `sort()` **方法**例如 `list1.sort()` 可直接使用。内存地址不变，其**修改原始列表，不返回排序结果**。

  所以不要写成 `print(list.sort())`。排序默认**升序**，使用 `sort(reverse=True)` 降序排序，或使用 `[::-1]`

- `sorted()` **函数**其不改变原有列表，**返回新的排序后的列表**
  例如 `list2=sorted(list1)`
- `reverse()` **逆序**不排序，和 `[::-1]` 效果一致

  数据量大的时候不要用，直接调用 `reverse`

- **索引方法**形如 `lst[1][0][2]` 同样需要注意越界问题

第三章　程序的控制结构

3.1　分支结构

3.1.1　双分支结构

- **紧凑格式** `< 表达式 1>if< 条件 >else< 表达式 2>`
  `print(' 猜 {} 了'.format(' 对'if a==9 else ' 错'))`

3.1.2　条件组合

- **优化原则**
  1. **与或非运算**时系统会有优化，如果 and 前已经算出 False 了，那么结果就是 False，后面的表达式计算机不会进行计算。
  2. 如果 or 前已经算出 True，那么结果就是 True，后面的表达式不计算
- **注意**
  1. 0，None 为假，**空对象为假**
     空对象包括 `""`, `()`, `[]`, `{}`
  2. **浮点数相减判断**两者差值小于一个足够小的数 `if abs(a-b)<=1e-8:`
  3. 使用 `pass` 可以跳过当前分支

3.2　异常处理

高级使用方式

```
try:
    <1> 正常执行的内容
except <异常类型>:
    <2>
else:
    <3> 没有异常时执行
finally:
    <4> 这一部分一定会被执行
```

常用于文件读写，并发任务时的资源访问等场景。
例如，完成读写文件后，要求关闭该文件。
若无错误，则执行 1→3→4，若报错，则执行 1→2→4

3.3 循环结构

- **遍历循环** For < 循环变量 > in < 遍历结构 >:
  < 语句块 >

  从结构里面按顺序取一个值赋给循环变量，语句的循环次数是固定的。

- **计数循环循环特定次**
  for i in range(N):
  < 语句块 >

  共循环 N 次，i 第一次为 0，最后为 N-1

- range(M,N,K) 从 M 到 N-1，步长为 K
- range(M) 从 0 到 M-1，一共 M 个数
- range(M,N) 从 M 到 N-1
- **无限循环** while < 条件 >: < 语句块 >
  反复执行语句块，直到条件不满足
- **循环控制保留字**
  - break 跳出并结束这个循环
  - continue 结束当前循环，继续执行后续次数循环
- **双重循环**

```
s="PYTHON"
while s!='':
    for c in s:
        print(c,end='')
    s=s[:-1]    #每次丢掉最后一个符号
```

得到 PYTHONPYTHOPYTHPYTPYP

第四章　元组集合字典

4.1　元组 tuple ( )

特点视作为只能读取数据，**不能修改数据的列表**

元组在处理大批量数据时，内存占用相比列表要小，且查找速度更快。
元组的括号可以省略例如 `tup="1",2,True`
`x1,x2,x3=tup` 自动分配 tup 中的内容给 x1,x2,x3，但需要注意数量左右一一对应

- **空元组** `tup=()`

  值得注意的是，若元组中仅有一个元素，需要在这个元素后**增加一个逗号**
  例如 `tup1=(50)` 和 `tup2=(50,)` 其类型不同

- **访问元组**按照下标或切片操作

- **修改元组 元组只读**。无法通过下标赋值，没有 replace 等方法。
  但其可以连接，例如 `tup1+ tup2` 得到一个新的元组；`tup1*3` 得到新的元组

- **删除元组**无法实现删除某个元素，可以用 `del` 删除整个元组

- **相关函数** `len()` `max()` `min()`

- `tuple()` 将某个东西强行转换为元组

- **封装和解封**自动加圆括号，自动去圆括号。
  `tup="1",2,True`

4.2　集合 set( )

4.2.1　基本内容

- **创建集合 仅使用** `set()`

```
num=set(["a","b","c"])
num=set("123456")
>>> {'1','4','6','2','3','5'}
```

- **特征**

  1. 自动去除所有的重复元素，每个集合**元素唯一**。因此可以**用其去除重复值**
     `lst=list(set(lst))` 注意，顺序可能改变

2. 集合内的元素是**无序**的，集合中元素的顺序可能与输入的顺序不同。因此不应该 使用下标索引集合。
3. 集合中可以包含类型不同的元素
4. **非空集合的判断**使用 len，长度为零

- **运算符**设 `a=set("abcd") b=set("bcd")`
  - `a-b` 差：a 去掉 ab 的交集
  - `a|b` 交集
  - `a&b` 并集
  - `a ^b` **对称差集**并集-交集 (**不同时包含**的元素)
  - `in / not in` 元素包含判断
  - `==` 判断两个集合是否全等
  - `!=` 判断两个集合不等
  - `> <` 检查包含关系（父级判断函数），此处为真子集
  - `>=` 子集

4.2.2 基本函数

- `len min max sum` 等
- `n.add(x)` **添加函数**, 向集合 n 中添加 x。若元素已存在，则不进行操作。
- `n.update(x)` 添加元素，参数可以是列表、元组、字典、集合等组合数学类型
  其不能直接添加单个值
- `n.discard(x)` **移除元素**，其不会报错。**其没有返回值**
- `n.pop()` 随机地删除一个集合元素。

  （实质上是删除第一个元素）
- `n.clear()` 清除集合中全部元素

4.3 字典 { }

字典由一系列**键-值**组成，每个键与一个值相关联，使用键来访问与之相关联的值。关联值可以是数字、字符串、列表甚至字典。
不要使用 `dict` 作为变量名。

4.3.1 字典创建

方法字典名 ={**键 1: 值 1**, **键 2: 值 2**,.....}

```
fruit_num={"apple":1,"orange":2,"banan":3}
```

注意

1. 同一字典中，键和值的类型**均可以不同**。

   ```
   any_type={"a":[1,2,3],2:8,True:{"hello"}}
   ```

2. 键和值均可以重复，但键重复，则仅最后一个有效。关键字必须唯一，且可哈希，可散列（例如列表）

3. **键必须不可变**。如 `["abc"]:2`、`{1:2}:5`、`(2,{}):6` 均错误，其包含有列表、字典等可变类型

4.3.2 元素读取

- **方法 1 值 = 字典名 [key]**

  使用 `fruit_num["apple"]` 获取值 1

  若键不存在，**则报错**。

- **方法 2** `值 = 字典名.get(key,default)`

  default 为 **key** 不存在时返回的默认值
  例如 `fruit_num.get("pear",-1) >>> -1`

4.3.3 元素新增修改

- **方法 1** `字典名 [key]= 新值`

  若 **key** 不存在，则直接添加新纪录。
  等同于新增记录。例如 `fruit_num["pear"]=4`

- **方法 2** `x1.update(x2)` 合并字典，如果关键字重复，则更新为 x2 中的新数据

4.3.4 元素删除

- `字典名.pop(key)` 根据关键字删除，同时删除键值。其**返回**被删除的键的对应值。

- `x.popitem()` 无需给 **key**，删除固定的最后一个。不确定被删元素，慎用。

4.3.5 列表遍历

- `n.items()` 所有的元素，**同时给出键值**。可以被转换为列表类型

  ```
  dict_items([('chinese', 80), ('python', 100), ('math', 59)])
  ```

- `n.keys()` 所有的**关键字**，仅给出 **keys**
  `dict_keys(['chinese', 'python', 'math'])`

- `n.values()` 所有的**数值**，仅给出 **values**

  ```
  dict_values([80, 100, 59])
  ```

- `for key,item in n.items():` 用于遍历字典 **n** 里面的 **key** 和 **item**

4.3.6 字典拷贝

- `copy()` 浅拷贝，例如 `b=a.copy()`
  如果字典里面的值带有列表，会存在问题
- `a=b` 直接赋值，即浅拷贝，引用对象（**同步修改**）

4.3.7 字典的特性

1. 列表动态数组，可变重设长度。元组静态数组不可变
2. 元组缓存于 Python 运行环境，每次使用无需访问内核分配内存
3. 字典的查找和插入**速度极快**，不会随着 **key** 的增加而变慢，但内存占用很大；列表查找插入时间随着元素的增加而增加，但占用空间小，内存少
4. 字典的 **key** 是不可变对象。

4.3.8 列表装配拆分

假设有两个列表：

```
>>> name=['zhang','zhao','wang']
>>> grades=[70,80,45]
```

装配起来使用 **zip 函数**：

```
>>> list(zip(name,grades))
[('zhang', 70), ('zhao', 80), ('wang', 45)]
```

假设拿到一个组合的列表：

```
new=[('zhang', 70), ('zhao', 80), ('wang', 45)]
```

仍然使用 **zip** 拆除，变量名前加 *：

```
>>> list(zip(*new))
[('zhang', 'zhao', 'wang'), (70, 80, 45)]
>>> a,b=zip(*new) 同时使用两个变量接收返回值
>>> a→  ('zhang', 'zhao', 'wang')
>>> b→  (70, 80, 45)
```

注：zip 只能使用一次，用完即释放原有内存。
枚举自动给元素序号，并以元组形式返回，当我们用下标时，可以以此查询：

```
>>> for i,v in enumerate(a):    print(i,v)
>>> 0 zhang    1 zhao     2 wang
```

第五章 函数

5.1 函数 Function

定义函数是一段功能模块，将常用的功能编写成函数利于代码复用。

程序设计思想：

1. 自顶向下逐步求精，顶层设计善于抽象（先不考虑细节，考虑总体功能分区实现），再进一步拆分，分而治之。
2. 模块化，结构化 → 函数；面向对象 → 类（变量 + 函数）

5.1.1 定义函数

```
def 函数名([传入参数变量列表]):  # []表示可选项
    函数体
    [return [返回值表达式]]
```

同时表示函数结束，若 return 无返回值，则返回空。

实参调用时给的**实际参数**

形参指的是定义时写的**形式参数**

函数调用函数名 (参数)

函数的定义必须出现在函数调用之前

5.1.2 函数参数

1. **位置参数**

   规则调用时与声明时一致，位置不能变更。形参和实参个数必须完全一致（不可省略），形参实参类型必须完全一致，否则均会报错

   示例 `def sum(begin,end,model):`

   `sum(1,10,0)`

2. **关键字参数**

   规则使用关键字参数传参，以 `name=value` 形式传递

   示例 `def sum(begin,end,model):`

   `sum(begin=1,end=10,model=0)`

   注意可以将关键字与位置参数混用，关键字参数必须在位置参数**之后**

   `sum(20,end=40,model=0)` 可行

   `sum(end=40,start=0,20)` 会报错

3. **默认参数**
   规则编写函数时可以给每个形参提供默认值。
   示例 `def sum(begin,end,model=0):`
   `sum(1,10)` 可省略最后一个
   注意形参书写中必须先列出没有默认值的参数

   ```
   def sum(begin=35,end,model=0):
   sum(50) 直接报错
   ```

4. **不定长参数**
   规则一个函数能够处理比声明时更多的参数，这些参数叫做不定长参数。

```
def 函数名([参数列表, ]*不定长参数,**不定长参数2):
```

示例

```
def print_info(arg1,*arg2):
    "打印任何传入的参数"
    print(arg1)
    for var in arg2:
        print(var)
    return
```

调用：`print_info(10,20,30)`

5.1.3 函数返回

- **函数的返回值** Python 在形式上允许返回多个值
- **返回一个值** `return A`
- **返回空值** `return` 等价于 `return None`
- **返回多个值** `return A,B`
- **返回大量数据**可利用列表、字典、元组等返回数据
  例如：`result.append(s)`
  `return result`

5.1.4 其他

- **嵌套调用**在一个函数里面调用其他函数

  通常程序定义一个包含程序主要功能的 main 的函数，使程序更规范易读。

示例

```
def sum(begin,end,model):
    sum=0
    for i in range(begin,end):
```

```
        if i%2==model:
            sum=sum+i
    return sum

def main():
    s=sum(1,10,0)
    print(s)

main()  # 函数调用语句
```

- **调用栈**原理：开辟内存栈空间，main 在底层，sum1 在顶部，后入先出。
- **递归调用**自己调用自己

  要求：先讨论分支、讨论清晰**结束递归的边界条件**，再找到需要递归的条件

  示例阶乘实现：

```
def fact(n):
    if n==1 or n==0:  #边界条件
        return 1
    return n*fact(n-1)
```

5.2 变量作用域

5.2.1 作用域类型

- **局部作用域 (Local)** 包含在 def 关键字定义的语句块中，不同函数内部的变量名与函数外的变量名为不同。函数为一个独立的内存空间。
  称为**局部变量**。
- **嵌套作用域 (Enclosing)** 函数中嵌套函数，上级影响下级
- **全局作用域 (Global)** 作用仅限于单个模块文件内
- **内置作用域 (Built-in)** 系统内固定模块里定义的变量，如 math 里的 pai
- **原则**局部变量不能扩散到全局

  全局变量能够渗透到全部

- **示例**

```
var=100  #全局作用域,覆盖全部范围
def fun1():
    var=200 #外部嵌套函数作用域
    print("var in fun1:",var)
    def fun2():  #嵌套定义
```

```
        var=300  #局部作用域
        # 只要在函数内赋值，就是局部
        print("var in fun2:",var)
        print("max in fun2",max)
        # max在内建函数作用域中，只读不能改变
    fun2()
fun1()
print("var in global",var)
```

输出：

```
>>> var in fun1: 200
>>> var in fun2: 300
>>> max in fun2 <built-in function max>
>>> var in global 100
```

5.2.2 作用域优先级

局部作用域 > 嵌套作用域 > 全局作用域 > 内置作用域

LEBG

错误案例

```
var=100
def fun():
    print(var)
    var=200  # 在此赋值导致变量类型无法确认
fun()
```

global 语句如果想要在函数内对全局变量赋值，需要使用 **global** 语句，例如：

```
var=100
def fun():
    global var
    print(var)
    var=200
fun()
```

在软件工程角度，不推荐直接修改全局变量。可以使用如下方法：

```
var=100
def fun(var):
    print(var)
    var=200  # 这里的修改不会改动外界的var
fun()
```

第六章　文件和数据库

6.1　文件概述

- **文件路径**关键属性：路径、文件名
- **根文件夹** Windows 中为 `D:\\`；Linux 中为 `/` 新加卷在 `/mnt` 文件夹下
- **绝对路径**总是从根文件夹开始
- **相对路径**指文件相对于当前工作目录所在的位置
- **示例**工作目录为 `"C:\Windows\System32"`

  若 `demo.txt` 位于这个 System32 文件夹下，则可表示为 `".\demo.txt"`
- **使用** `.\` 表示当前目录；`..\` 表示所在目录的父级目录
- **字符编码**汉字一般使用 GBK，**UTF-8**（世界通用）

6.2　文件操作

6.2.1　文件的打开

```
1 file = open(file_name,mode="r",buffering=-1,encoding=None)
```

- **mode** r 只读（默认） **w** 覆盖 a 写入追加 **b** 以二进制格式操作
  `r+` 从开头写入新的数据，会覆盖原有内容
  `rb` 以二进制只读格式打开文件
  `rb+` 通常针对非文本文件
  `w` 打开时即清空文件中原有内容
  `w+` 对该文件拥有读写权限
- **buffering** 表示是否使用缓冲区
- **注意**该函数仅获得文件句柄，即文件处理权限没有进行读数据的操作，未复制到内存
- **示例**例如 `file=open("./a.txt") print(file)`
  输出相关文件信息：
  `<_io.TextIOWrapper name='a.txt' mode='r' encoding='cp936'>`

6.2.2 文件对象的属性

- `file.name` 返回文件名称
- `file.mode` 返回文件打开的模式
- `file.encoding` 返回文件使用的编码格式
- `file.closed` 判断文件是否已关闭

6.2.3 文件的关闭

- `file.close()` 使用 open() 打开的文件对象，**必须手动进行关闭**
- **注意**在向文本格式打开的文件中写入数据时，只有使用 close() 函数关闭文件时，才会将缓冲区中的数据真正写入文件中。写数据时，会先临时存储到缓冲区。

  可以使用 `file.flush()` 方法可强制存盘
- `with open() as f:` 使用此法可在 with as 语句执行完毕后自动关闭已打开的文件

6.2.4 读文件

- `read()` **逐个字节**或字符读取文件内容。

  将文件的全部内容读出
  可使用 size 参数设置每次读取的最大字节
  `f.read(6)` 读取前 6 个字符
- `readline()` **逐行读取**文件中内容。以 `\n` 作为读取一行的标志，会包含最后的换行符 `\n`，再加上 print 输出内容时默认会换行

  `f.readline([size])` 其中 size 指读取每一行时，一次最多读取的字符数
- `readlines()` 一次性**读取多行内容**，返回一个字符串列表，每个元素为每行内容 (包含结尾换行符)

  不可读取全部文件，最后会报错。
  可用于读完一行打印一行：
- 读取特定行数的函数：

```python
def realine(filename,lines,encode="utf-8"):
    with open(filename,encoding=encode) as file:
        count=0
        for line in file:
            newline=line.strip()
            if len(newline)==0 or newline[0]=="#":
                continue
            print(newline)
            count+=1
```

```
            if count>=lines:
                break
realine("a.txt",5)
```

- **收集操作系统的系统配置信息** psutil 模块

6.2.5 写文件

- `write()` 格式：**file.write(string)**

  必须保证文件打开模式支持写入。写入数据时不会换行，需要手动输入换行符。
  例如：`f=open('a.txt','a')`
  `f.write('\n 新数据') f.close()`

- `writelines()` 将**字符串列表**写入到文件中

  可以自动换行
  可轻松地将 A 文件的数据复制到 B 文件中：

```
f=open('A.txt','r')
n=open('B.txt','w+')
n.writelines(f.readlines())
f.close()
n.close()
f.writelines(["123","456"])
```

6.2.6 文件指针 pointer

- **概念**用于标明文件读写的起始位置。可实现读取文件中指定位置的数据。

- `tell()` 判断当前文件指针所在的位置。

  若全为字节，则按照字节；若中英文混合，则按照字符来。
  格式：**file.tell()** 初始为 0
  例如：读取三个字节 `f.read(3)` 此时若继续 `f.read()` 则输出未被读过的所有数据

- `seek()` 移动文件指针到指定位置
  格式：**file.seek(offset[,whence])**
  whence 可选，指定文件指针要放置的位置

  0 文件头（默认），1 当前位置，2 文件尾

  offset 表示相对于 whence 位置的偏移量（字节数），正为向后，负为向前
  `seek(3,0)` 移动到距开头处 3 个字节的位置
  `seek(5,1)` 移动到距当前位置 5 字节后

  当 offset 非 0 时，要求文件**必须以二进制格式打开**，否则报错。即使用 `'rb'`、`'ab'`、`'wb'`

- **示例程序：**输出最后 5 行数据

```
with open("D:/Temp/test.txt",mode="rb") as f:
    f.seek(-150,2)  # 设置一个大的数字
    for row in f.readlines()[-5:]:  # 切片最后五行
        print(row.strip().decode("utf-8"))
        # 去掉空格换行等后把编码改变
```

6.2.7 文件系统相关命令

- **路径拼接** `os.path.join(path,name)` `os.sep.join(path,name)`

  其可以自动拼接并返回文件路径字符串，并包含正确的路径分隔符

  `import os`

  `os.path.join("demo","path.txt")`

- **os 中的一些小方法**
  - `os.getcwd()` 获取当前工作路径的字符串
  - `os.chdir("path")` 可改变目录
  - `os.name` 获取 os 名称
  - `os.system()` 执行 shell 命令
  - `os.getenv()` 获取环境变量，例如 `os.getenv("path")`
  - `os.remove()` 移动文件
  - `os.path.getsize(path)` 返回文件大小、若文件不存在报错
  - `os.path.exists(path)` 判断文件是否存在
  - `os.path.isfile(path)` 判断是否为文件
  - `os.path.split()` 路径拆分为路径名和文件名（以元组形式）
  - `os.walk(path)` 遍历文件路径，返回三元组 `(root,dirs,files)`
- **sys 模块与提取命令行参数**

  可提供操作系统的属性信息

  `sys.version` 获取 python 解释程序的版本信息

  `sys.platform` 提供操作系统信息

第七章　面向对象程序设计

结构化程序设计 自顶向下逐步求精，将各种功能分解出来
把复杂问题逐步分解为易于理解实现的一系列功能模块**【分治】**，主要由函数实现。
Google：map reduce 架构/ Python 中有 `map()` `reduce()` 函数实现数据并行计算
面向对象程序设计
封装：把方法、属性、事件集中到一个**统一的类中**，对使用者屏蔽其中细节（有了外壳）
继承：在被继承的类 (父类、基类、超类) 基础上，进行扩展生成新的类-**子类**。
多态性：同样的函数对于不同的对象具有不同的实现。
例如 + 号，对 `3+4`、`"3"+"4"` 有不同的实现
优势：提高代码复用性、使编码灵活可维护、提高可扩展性、提高开发效率

7.1　类 Class

包含有**属性** (→ 变量)、**方法** (→ 函数)
概念描述具有相同属性和方法的**对象的集合**，因此是抽象的。定义了该集合中每个对象所共有的属性和方法。
例如类：学生，每个学生都有姓名、性别、学号等
实例化 instance 创建一个类的**具体**实例-**类对象**。变化的这个动作叫做实例化。变化的结果叫对象，故对象有时也称为实例。一个类可以有众多的实例。
方法类中定义的函数
属性类变量，成员变量。
定义语法 `class Name():` 一般括号内为空，若存在 object，表示从 object 中继承而来
初始化语法 `def __init__(self,x,y,z,.....):` 也称为 x,y,z, 为希望传参进入的参量
第一个参数为 self，无需传参，表示对象自己
析构方法 `def __del__(self):` 用于销毁对象本身，释放对象所用的内存

> python 有优秀的内存回收机制，一般不需要程序员重新定义或手动调用析构方法。当程序运行结束，python 会自动触发 `__del__()` 方法

示例

```
class Myclass:
    var=1

# 一个关于圆形的类
class Circle(object):
```

```
    def __init__(self,radius):  # 初始化，一个构造方法__init__，用来构造对象；self就是对象自己
        self.radius=radius      # 对象属性，仅在类里面使用
    def area(self):             # 类方法，每个方法的第一个形参都是self
        return 3.14*(self.radius)**2    # 求圆面积

one_circle=Circle(2)            # 声明类对象，实例化
print(one_circle.area())
```

7.2 对象 Object

概述在开发复杂庞大的程序时，需要有丰富的需求，同时要求开发简便易于维护，即智能化地实现。

对象是一个整体，一切皆对象。具备了属性 (n.) 和功能 (v.) 的就是对象。

变量类型、字符串、函数、面向对象也是对象。

> 开发的开始为设计需求，先考虑属性与功能，随后再思考编程实现。此时设计程序不以变量来设计（零碎的）。在计算机中，object 实现为 class（类）。
>
> 对象和类是一个对应的概念，类 是**抽象的**，对象 是非常**具体的**。
>
> `"a".count("a")` 中的点 • 即代表调用对象的属性或是功能。如果有 ( ) 则为方法，没有括号则为属性。例如 `arr.shape` 返回具体的属性值。

创建对象即类的实例化

例：学生对象创建

```
class Student():      # 初始化代码不必要
    def set_id(self,x):  # 设置属性ID值，self表示对像自己，x是传进来的参数
        self.id=x     # 表示对象自身的ID，本质上类似变量，就是属性
    def show_id(self):
        print(self.id)

stu1=Student()    # 声明了具体的学生对象，对象实例化。即创建类对象
stu1.set_id(12345)        # 调用类里面的方法
stu1.show_id()

stu2=Student()
print(isinstance(stu2,Student))  # 判断stu2是否为Student的实例
```

7.3 属性和方法

7.3.1 类属性和实例属性

- **类属性**在类的名称下直接写变量，在各个方法之外。可以通过类的名称打点直接访问。
  例如 `Student.school`
- **对象属性**形如 `self.aaa`，可以在其他方法中直接使用（方法内共享）

```
class Student:
    school="某大学"      # 类属性
    def set_id(self,x):  # x就是普通局部变量
        self.id=x      # 对象属性，可以在其他方法中直接使用（方法内共享）
        y=0              # 一个方法内的局部变量
    def show_id():
        print(self.id)

Student.school()          # 调用类属性
stu3=Student()
stu3.school()             # 可以通过实例访问类的属性
```

7.3.2 类的方法

对象方法类对象使用的方法，方法名的第一个形参是 self
调用时由类对象来调用，**不可通过类名称直接调用 (如 `Student.show_id()` 报错)**
但可以 `Student.show_id(stu5)`（传一个类对象）

```
def show_id():                # 对象方法
    print(self.id)       # 可直接调用 Stu5.show_id() 使用类对象访问
```

7.4 封装与访问控制

封装把数据和操作方法的细节隐藏，让调用的人方便调用。

把一组有密切联系的数据和操作打包成一个不可分割的整体；对外只提供专门的接口（方法）供使用者访问部分数据和操作

访问控制

- **private 私有**仅可通过方法对对象属性进行赋值，不对外公开**【命名时在名称左侧加双下划线】**
- **public 公有**可直接赋值，完全公开**【默认】**
- **proteted 保护**对外公开一部分**【在左侧加一个下划线】**，不允许使用 import 导入，即跨文件使用。

```
def set_id(self,x):  # x就是普通局部变量
    self.__id=x  # 创建私有对象
    self._id=x   # 创建保护对象
    y=0          # 一个方法内的局部变量
```

还可以创建公有或私有方法，可使用公有方法调用私有方法。

7.5 继承和多态

继承提高代码复用率。以原有的类（基类、父类）为基础，派生出新的类（派生类、子类）

1. 子类继承了父类的数据和方法，可以定义新的方法，也允许重新定义一个旧的方法，为其增加新的特性。子类不能直接访问父类中的私有方法和私有属性。
2. 派生类同样可以作为父类，再派生出新的子类，形成类的层次结构（UML 图）。
3. 如果不写继承，如 `class pa:`，所有类似的均从抽象类 object 继承，等价于 `class pa(object):`
4. 构造方法内声明的变量**不会被自动继承**。继承方法：**super().___init___( )** 必须手动**显式地**调用父类构造方法。等价于 `Student.__init__(self,.....)`

单继承一个派生类只有一个直接积累
多继承一个派生类同时有多个基类（最好不要用多继承）
语法结构 `class 子类名（父类名 1，父类名 2，.....):`
`def __init__(self，父类参量，新参量):`
`supper().__init__(父类参量)`
`# 类定义部分`
覆盖 Override. 属于继承的一种。不改变名字，改变动作。

```
class ParentClass:
    def show_info(self,info):
        print("A")

class ChildClass(ParentClass):
    def show_info(self,info):
        print("B")
    def parent_show_info(self,info):     # 调用父类方法
        ParentClass.show_info(self,info)

obj=ChildClass()
obj.show_info()
obj.parent_show_info()
```

多态：根据外界的环境灵活变化自身功能

第八章　模块和包

8.1　模块 modules

模块将代码存储为 python 文件，都称之为模块。每一个 python 脚本文件也为模块（a.py）
包含多个函数与类
导入模块 import 语句

> **模块导入仅发生一次，放在文件开头**
> **方法一 import 模块名 1 [as 别名 1], 模块名 2, [as 别名 2], ...**
> 导入全部成员，消耗内存大，调用成员时需要用模块名作为前缀，否则报错
> 例如 `import math as m`
> **方法二 from 模块名 import 成员名 [as 别名 1], ...**
> 只会导入模块中的指定成员，调用时无需附加前缀
> `from math import pi,e,sqrt`
> `from math import *` 可以导入 math 中全部成员，无需附加前缀了

查看模块 `dir()` 查看指定模块包含的全部成员

8.2　包 package

包将多个功能模块存储在同一文件夹下，作为整体打包处理。文件夹下通常存在 **___init___.py** 的初始化文件（文件一般为空），作用为告诉 python 将此文件夹按包处理。
包名即为文件夹名
包含多个模块
导入包 from 文件夹名. 模块名 import 成员名
存在限制

8.3　常用内置模块

- `math` 提供数学运算相关函数
- `random` 随机数函数
- `datetime` 提供日期和事件相关函数
- `logging` 提供日志相关函数
- `re` 提供正则表达式相关函数

- os 提供文件操作相关
- sys 操作系统相关信息函数
- time 提供时间戳，格式化时间

8.4 math 模块

- 常量 pi e
- 函数 fabs 返回绝对值
- factorial(a) 阶乘
- gcd(a,b) 最大公约数
- 幂函数 log(a,b) 以 b 为底的对数
  log2(a) 以 2 为底
  log1p(a) 以 e 为底
  log10(a) 以 10 为底
  sqrt(a) 平方根
  pow(a,b) a 的 b 次幂
- 三角 degrees(a) 返回弧度 a 对应的角度
  radians(a) 返回角度 a 对应弧度

8.5 random 模块

注意：生成的是伪随机数

获取一个随机数

- random() 返回 0-1 之间的随机浮点数
- randrange(start,stop,step) 返回 start 到 stop，步长为 step 的随机整数
- randint(a,b) 返回 a 和 b 之间的随机整数，一次一个
  随机数可以等于起始值，但不会等于终止值。

获取一个随机序列

更多的像统计中的**抽样**

- choice(seq) 返回序列中随即一个数
- choices(seq,k) 随机选择 k 个序列中的数，得到一个新序列，可以重复（有放回）
- sample(seq,k) 随机选择 k 个序列中的数，得到一个新序列，不会重复（无放回）
- shuffle(seq) 随机打乱序列中的数

8.6 datetime 模块

采用类似于字典的形式区分，

类

- `date` 表示日期
- `time` 时间
- `datetime` 日期时间
- `timedelta` 时间跨度
- `tzinfo` 表示时区

常量

- `MINYEAR` 最小允许年份，为 1
- `MAXYEAR` 最大，9999

方法

- `data(year,month,day)` 创建一个 date 对象
- `today()` 返回今天的 date
- `now()` 返回详细的 datetime 对象
- `weekday()` 返回今天是周几
- `strgtime()` 格式化 datetime 对象

格式化字符串

- `%Y` 四位年
- `%y` 两位年
- `%m` 月

8.7 time 模块

`time()` 提取当前时间戳