

#coding=utf-8

# builtin\_function.py 内置函数

import os

def fun():

  all([True, False]) # 迭代器(为空or)所有元素为true,返回true => False

  any([True, False]) # 迭代器任意一个元素为true,返回true => True

  num = abs(-1.23) # 绝对值

  num = pow(5, 3) # 幂次方 x\*\*y =>125

  num = pow(5, 3, 3) # 幂次方再取余 (x\*\*y) % z => 2

  num = round(1.23) # 返回浮点数近似值,默认保留0位,且四舍五入

  num = round(1.23, 5) # 5为保留小数点后位数

  num = sum([1, 2, 3, 4, 5]) # 对列表数字求和

  strs = ascii(os) # 返回对象的ascii字符串形式

  strs = bin(123) # 整数转为二进制字符串(若被转对象非int类型可在\_\_index\_\_里定义)

  boolean = bool(0) # 获取真假(假:None / False / 任何类型的0 / 空""()[]{} / \_\_bool\_\_返回False / \_\_len\_\_返回0)

  bytes = bytearray("You were, are and will be in my heart!", "utf-8") # 字符串(字符串,编码):按编码转为字节序列 / 数字(数字):生成相应大小的空数组 / 无参():0数组

  bytes = bytes("You were, are and will be in my heart!", "utf-8") # 同bytearray(),但是不可变

  strs = chr(123) # 整数(Unicode)转为char类型(范围[0, 1 114 111])

  num = ord("{") # char类型转为整数(Unicode)

  num = complex("1+2j") # 将字符串或数字转为复数(不能含有空格)

  num = float('1.1') # 构建浮点数

  strs = format(123,'f') # 格式化(类型:x,b,f... / 位移:>(>10),<,^ / 长度:5 / E)

  strs = hex(123) # 将整数转为16进制字符串

  strs = oct(123) # 十进制转成8进制字符串

  num = int(123.1) # 转为整数

  strs = str(123) # 将对象转为字符串

  strs = str(b"123", "utf-8")

  elem = max([1,2,3,4], [2,4,6]) # 可接收1个iterable,或多个元素 => [2, 4, 6]

  elem = max(1,2,3,4) # => 4

  elem = max([1,2,3,4, 6], key = lambda x : x == 2) # => 2

  elem = min(1,2,3,4) # 与max相反

  exe = compile("print('O\_O')", filename = 'strs', mode='exec') # 编译为代码, mode(编译模式:可执行语句'exec',单个语句'eval',交互式语句'single')

  exec(exe) # 执行已编译代码

  exec("print('O\_O')") # 执行String未编译代码

  eval(compile("print('O\_O')", filename = 'strs', mode='eval')) # 不接受字符串类型

  delattr(Clazz("Hello!"), "name") # 删除某对象属性 = del clazz.name 详情见 类 文章的 反射 代码块(//www.jb51.net/article/128897.htm)

  setattr(Clazz("Hello!"), "name", "World!") # 给对象某属性赋值

  strs = getattr(Clazz("Hello!"), "name") # 获取某个对象的属性值

  strs = getattr(Clazz("Hello!"), "name", -1) # -1为未找到属性的返回

  boolean = hasattr(Clazz("Hello!"), "name") # 该对象是否有该属性

  lists = dir(Clazz("Hello!")) # 查看函数

  clazz = type(os) # 获取类型

  dicts = globals() # 获取当前全局函数与对象

  dicts = locals() # 获取当前局部函数与对象

  dicts = vars() # 同locals() (\_\_dict\_\_)

  dicts = vars(Clazz("Hello!"))

  num = hash(Clazz("Hello!")) # 获取某对象的hash值

  help(os) # 获取某对象的帮助文档

  num = id(os) # 获取某对象的id

  boolean = isinstance(Clazz("Hello!"), Clazz) # 对象是否是该类的实例

  boolean = issubclass(Clazz, Clazz) # 该类(前)是否是该类(后)的子类(包括自己)

  strs = repr(os) # 将对象转为字符串表达形式

  # - super() # 代理父类对象,详情见 类 文章(//www.jb51.net/article/88315.htm)

  # - memoryview(obj) # 内存视图,详解 数据结构 文章(//www.jb51.net/article/128892.htm)

  dics = dict() # 创建字典

  tups = divmod(10, 5) # 返回元组,(商(10/5),余数(10%5))

  lists = enumerate(['a', 'b', 'c']) # 返回枚举对象

  lists = filter(lambda x: True if (ord(x) > 66) else False, ['a', 'b', 'c']) # function为Frue保留,False移除

  sets = frozenset([1, 2, 3]) # 返回新的frozenset对象(集合)

  num = len([1, 2, 3]) # 长度

  lists = list((1, 'a')) # 转为list类型

  tups = tuple([1, 2, 3]) # 转为tuple元组类型

  ran = range(5) # 不可变序列

  ran = range(0,5)

  ran = range(0,5,2) # (起始,结束,增加量)

  sets = set([1,2,3]) # 返回set集合

  maps = map(lambda x, y: x\*y, [1, 2, 3], [65, 66, 67, 68]) # 返回一个迭代器,元素通过自定义函数筛选,可接收多个iterable参数 => [65, 132, 201]

  iters = zip(["A", "B", "C", "D", "E", "F"], [1, 2, 3]) # 创建新的迭代器, 聚合每个迭代器元素 => [('A', 1), ('B', 2), ('C', 3)]

  iters = iter([1, 2, 3, 4, 5]) # 返回一个迭代器对象

  elem = next(iters) # 从迭代器中获取下个元素; 实现原理见 内置函数 文章 lis 块代码(//www.jb51.net/article/128890.htm)

  iters = reversed([1,2,3]) # 返回反向的遍历器 => [3, 2, 1]

  lists = [1,2,3,4,5][slice(3)] # 切片 => [1, 2, 3]

  lists = [1,2,3,4,5][slice(1,3)] # [slice(3) == slice(None, 3, None) / slice(1,3) == slice(1, 3, None) / slice(1,3,1) == slice(1, 3, 1)

  lists = sorted([2,5,3,1,4]) # 排序 => [1, 2, 3, 4, 5]

  lists = sorted(['a','B',';','t','D','1'], key = lambda x : ord(x), reverse = True) # key:比较键的函数, reverse是否反向遍历

  strs = input("请输入数据:") # 输入数据

  f = open("temp.txt", "r+") # 打开文件,详情见os文章

  print("字符串%d"%123) # 打印字符 => 字符串123

  print("字","符", "串", sep="-") # sep为分隔 => 字-符-串

  print("字","符", "串", sep="-", end="\r\n") # end为尾部 => 字-符-串/r/n

  print("字","符", "串", sep="-", end="\r\n", file=open("temp.txt","w+")) # 打印到文件

class Clazz:

  def \_\_init\_\_(self, name):

    self.name = name;

  @classmethod # 将函数包装成类方法

  def setName\_cls(cls, name):

    pass

  @staticmethod # 将函数包装成静态方法

  def setName\_sta(name):

    pass

  def getname(self):

    return self.name

  def setname(self, value):

    self.name = value

  def delname(self):

    del self.name

  # property(fget=None, fset=None, fdel=None, doc=None) # 返回一个property 属性

  # property 为属性方法, 有两种实现方式,详情见 类 文章的 属性方法代码块(//www.jb51.net/article/68235.htm)

  x = property(getname, setname, delname)

if \_\_name\_\_ == "\_\_main\_\_":

  fun()

  # property 的使用

  c = Clazz("柳岩")

  print(c.x) # => 柳岩

  c.x = '汤唯'

  print(c.getname()) # => 汤唯

  del c.x

dict 语法：

class dict(\*\*kwarg)

class dict(mapping, \*\*kwarg)

class dict(iterable, \*\*kwarg)

参数说明：

\*\*kwargs -- 关键字

mapping -- 元素的容器。

iterable -- 可迭代对象。

返回值

返回一个字典。

'''

print(dict()) #创建空字典

dict1=dict(a='a', b='b', t='t') #传入关键字 构建字典

print(dict1) #输出 {'a': 'a', 'b': 'b', 't': 't'}

dict2=dict(zip(['one', 'two', 'three'], [1, 2, 3])) # 映射函数方式来构造字典

print(dict2) #输出 {'one': 1, 'two': 2, 'three': 3}

dict3=dict([('one', 1), ('two', 2), ('three', 3)]) # 可迭代对象方式来构造字典

print(dict3) #输出 {'one': 1, 'two': 2, 'three': 3}

15.dir()函数

'''

dir() 函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。如果参数包含方法\_\_dir\_\_()，该方法将被调用。如果参数不包含\_\_dir\_\_()，该方法将最大限度地收集参数信息。

语法

dir 语法：

dir([object])

参数说明：

object -- 对象、变量、类型。

返回值

返回模块的属性列表。

'''

print(dir()) #获得当前模块的属性列表

#输出 ['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_']

print(dir([])) #获得列表的方法

#输出 ['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_

print(dir(str)) #获得字符串的方法

#输出 ['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

print(dir(dict)) #获得字典的方法

#输出 ['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

def update\_func(var):

print("var 的内存地址:", id(var))

var += [4]

lst\_1 = [1, 2, 3]

print(dir()) #输出 ['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'lst\_1', 'update\_func']

class Shape:

def \_\_dir\_\_(self):

return ['area', 'perimeter', 'location']

s = Shape()

print(dir(s)) #输出 ['area', 'location', 'perimeter']

#参考网页 https://www.yuzhi100.com/tutorial/python3/python3-neizhihanshu-dir

16.divmod()函数

'''

python divmod() 函数把除数和余数运算结果结合起来，返回一个包含商和余数的元组(a // b, a % b)。

在 python 2.3 版本之前不允许处理复数。

函数语法

divmod(a, b)

参数说明：

a: 数字

b: 数字

'''

print(divmod(20,4)) #返回 (5, 0)

print(divmod(7,2)) #返回 (3, 1)

print(divmod(8,2)) #返回 (4, 0)

#print(divmod(1+2j,1+0.5j))

#报错 TypeError: can't take floor or mod of complex number.

17.enumerate()函数

'''

enumerate是翻译过来是枚举的意思，看下它的方法原型：

enumerate(sequence, start=0)，返回一个枚举对象。

sequence必须是序列或迭代器iterator，或者支持迭代的对象。

enumerate()返回对象的每个元素都是一个元组，

每个元组包括两个值，一个是计数，一个是sequence的值，

计数是从start开始的，start默认为0。

---------------------

'''

a=["q","w","e","r"]

c=enumerate(a)

for i in c:

print(i)

'''

输出如下:

(0, 'q')

(1, 'w')

(2, 'e')

(3, 'r')

'''

a=["w","a","s","d"]

#这里加了个参数2，代表的是start的值

c=enumerate(a,2)

for i in c:

print(i)

'''

输出如下：

(2, 'w')

(3, 'a')

(4, 's')

(5, 'd')

'''

a=["q","w","e","r"]

#创建一个空字典

b=dict()

#这里i表示的是索引，item表示的是它的值

for i,item in enumerate(a):

b[i]=item

print(b) #输出 {0: 'q', 1: 'w', 2: 'e', 3: 'r'}

for i,j in enumerate('abc'):

print(i,j)

#输出结果

# 0 a

# 1 b

# 2 c

18.eval()函数

'''

eval() 函数用来执行一个字符串表达式，并返回表达式的值。

语法

以下是 eval() 方法的语法:

eval(expression[, globals[, locals]])

参数

expression -- 表达式。

globals -- 变量作用域，全局命名空间，如果被提供，则必须是一个字典对象。

locals -- 变量作用域，局部命名空间，如果被提供，可以是任何映射对象。

返回值

返回表达式计算结果。

'''

x=7

print(eval('3\*x')) #返回 21

print(eval('pow(2,2)')) #返回 4

print(eval('3+5')) #返回 8

#eval函数还可以实现list、dict、tuple与str之间的转化

#1.字符串转换成列表

a = "[[1,2], [3,4], [5,6], [7,8], [9,0]]"

print(type(a)) #返回 <class 'str'>

b = eval(a)

print(type(b)) #返回 <class 'list'>

print(b) #输出 [[1, 2], [3, 4], [5, 6], [7, 8], [9, 0]]

#2.字符串转换成字典

a = "{1: 'a', 2: 'b'}"

print(type(a)) #返回 <class 'str'>

b = eval(a)

print(type(b)) #返回 <class 'dict'>

print(b) #输出 {1: 'a', 2: 'b'}

#3.字符串转换成元组

a = "([1,2], [3,4], [5,6], [7,8], (9,0))"

print(type(a)) #返回 <class 'str'>

b=eval(a)

print(type(b)) #返回 <class 'tuple'>

print(b) #输出 ([1, 2], [3, 4], [5, 6], [7, 8], (9, 0))

19.exec()函数

'''

函数的作用：

动态执行python代码。也就是说exec可以执行复杂的python代码，而不像eval函数那样只能计算一个表达式的值。

exec(source, globals=None, locals=None, /)

source：必选参数，表示需要被指定的python代码。它必须是字符串或code对象。如果source是一个字符串，该字符串会先被解析为一组python语句，然后执行。如果source是一个code对象，那么它只是被简单的执行。

返回值：

exec函数的返回值永远为None。

eval()函数和exec()函数的区别：

eval()函数只能计算单个表达式的值，而exec()函数可以动态运行代码段。

eval()函数可以有返回值，而exec()函数返回值永远为None。

'''

x = 10

def func():

y = 20

a = exec("x+y")

print("a:",a) #输出 a: None

b = exec("x+y",{"x":1,"y":2})

print("b:",b) #输出 b: None

c = exec("x+y",{"x":1,"y":2},{"y":3,"z":4})

print("c:",c) #输出 c: None

d = exec("print(x,y)")

print("d:",d) #输出 d: None

func()

x = 10

expr = """

z = 30

sum = x + y + z #一大包代码

print(sum)

"""

def func():

y = 20

exec(expr) #10+20+30 输出60

exec(expr,{'x':1,'y':2}) #30+1+2 输出 33

exec(expr,{'x':1,'y':2},{'y':3,'z':4}) #30+1+3，x是定义全局变量1，y是局部变量 输出34

func()

#参考原博客 https://www.cnblogs.com/yangmingxianshen/p/7810496.html

20.filter()函数

'''

filter() 函数是一个对于可迭代对象的过滤器，过滤掉不符合条件的元素，

返回的是一个迭代器，如果要转换为列表，可以使用 list() 来转换。

该函数接收两个参数，第一个为函数的引用或者None，第二个为可迭代对象，

可迭代对象中的每个元素作为参数传递给函数进行判，然后返回 True 或 False，最后将返回 True 的元素放到迭代器中

下面看下fiter()的用法：

'''

my\_list=[1,2,'',3,4,'6',' ']

new\_list=list(filter(None,my\_list))

print(new\_list)

#None 函数 过滤掉'' 而不是过滤掉空字符串

def is\_oushu(x):

return x%2==0

new\_list=list(filter(is\_oushu,list(range(1,11))))

print(new\_list)

#过滤掉不是偶数的数

a=[1,2,3,4,5,6,2,2,2,]

print(list(filter(lambda x:x!=2,a)))

#筛选出列表里所有的不是 2 的元素

21.float()函数

'''

描述

float() 函数用于将整数和字符串转换成浮点数。

语法

float()方法语法：

class float([x])

参数

x -- 整数或字符串

返回值

返回浮点数。

'''

print(float(1)) #输出 1.0

print(float(112.0)) #输出 112.0

print(float('123')) #输出 123.0

print(float(True)) #输出 1.0

print(float(False)) #输出 0.0

#print(float('a'))

#报错 ValueError: could not convert string to float: 'a'

#可以参考博客 https://blog.csdn.net/TCatTime/article/details/82932818

22.format()函数

'''

自python2.6开始，新增了一种格式化字符串的函数str.format()，此函数可以快速处理各种字符串。

语法

它通过{}和:来代替%。

请看下面的示例，基本上总结了format函数在python的中所有用法

'''

#通过位置

print ('{0},{1}'.format('chuhao',20))

#chuhao,20

print ('{},{}'.format('chuhao',20))

#chuhao,20

print ('{1},{0},{1}'.format('chuhao',20))

#20,chuhao,20

#通过关键字参数

print ('{name},{age}'.format(age=18,name='chuhao'))

#chuhao,18

class Person:

def \_\_init\_\_(self,name,age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return 'This guy is {self.name},is {self.age} old'.format(self=self)

print (str(Person('chuhao',18))) #This guy is chuhao,is 18 old

#通过映射 list

a\_list = ['chuhao',20,'china']

print ('my name is {0[0]},from {0[2]},age is {0[1]}'.format(a\_list))

#my name is chuhao,from china,age is 20

#通过映射 dict

b\_dict = {'name':'chuhao','age':20,'province':'shanxi'}

print ('my name is {name}, age is {age},from {province}'.format(\*\*b\_dict))

#my name is chuhao, age is 20,from shanxi

#填充与对齐

print ('{:>8}'.format('189'))

# 189

print ('{:0>8}'.format('189'))

#00000189

print ('{:a>8}'.format('189'))

#aaaaa189

#精度与类型f

#保留两位小数

print ('{:.2f}'.format(321.33345))

#321.33

#用来做金额的千位分隔符

print ('{:,}'.format(1234567890))

#1,234,567,890

#其他类型 主要就是进制了，b、d、o、x分别是二进制、十进制、八进制、十六进制。

print ('{:b}'.format(18)) #二进制 10010

print ('{:d}'.format(18)) #十进制 18

print ('{:o}'.format(18)) #八进制 22

print ('{:x}'.format(18)) #十六进制12

#参考博客 https://www.cnblogs.com/gide/p/6955895.html

23.frozenset()函数

'''

描述

frozenset() 返回一个冻结的集合，冻结后集合不能再添加或删除任何元素。

语法

frozenset() 函数语法：

class frozenset([iterable])

参数

iterable -- 可迭代的对象，比如列表、字典、元组等等。

返回值

返回新的 frozenset 对象，如果不提供任何参数，默认会生成空集合。

'''

a=frozenset(range(10))

print(a)

#输出 frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})

b=frozenset('ltftyut1234')

print(b)

#输出 frozenset({'2', '4', 't', 'f', '1', '3', 'l', 'y', 'u'})

# 1 frozenset([iterable])

# 2 set和frozenset最本质的区别是前者是可变的，后者是不可变的。当集合对象会被改变时（例如删除，添加元素），只能使用set，

# 3 一般来说使用fronzet的地方都可以使用set。

# 4 参数iterable：可迭代对象。

24.getattr()函数

'''

描述：

getattr()函数用于返回一个对象属性值

语法：

getattr(object,name,default)

参数：

object--对象

name--字符串，对象属性

default--默认返回值，如果不提供该参数，在没有对应属性时，将触发AttributeError。

返回值：

返回对象属性值

'''

class People():

sex='男'

def \_\_init\_\_(self,name):

self.name=name

def peopleinfo(self):

print('欢迎%s访问'%self.name)

obj=getattr(People,'sex')

print(obj) #返回值 男

#obj=getattr(People,'sexage')

#print(obj)

'''

报错。。。

Traceback (most recent call last):

File "G:/pythonAI/Python\_funs/getattr函数详解.py", line 24, in <module>

obj=getattr(People,'sexage')

AttributeError: type object 'People' has no attribute 'sexage'

'''

obj=getattr(People,'sexage',None)

print(obj) #返回值 None

25.globals() 函数

'''

描述

globals() 函数会以字典类型返回当前位置的全部全局变量。

语法

globals() 函数语法：

globals()

参数

无

返回值

返回全局变量的字典

'''

a='ltftyut1234'

print(globals()) # globals 函数返回一个全局变量的字典，包括所有导入的变量。

# {'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': '\n描述\nglobals() 函数会以字典类型返回当前位置的全部全局变量。\n语法\nglobals() 函数语法：\nglobals()\n参数\n无\n返回值\n返回全局变量的字典\n', '\_\_package\_\_': None, '\_\_loader\_\_': <\_frozen\_importlib\_external.SourceFileLoader object at 0x000001C5A50FB4E0>, '\_\_spec\_\_': None, '\_\_annotations\_\_': {}, '\_\_builtins\_\_': <module 'builtins' (built-in)>, '\_\_file\_\_': 'G:/pythonAI/Python\_funs/globals函数详解.py', '\_\_cached\_\_': None, 'a': 'ltftyut1234'}

def zero\_promo():

return 0

def one\_promo():

return 1

def two\_promo():

return 2

def hello():

print("Hello")

if \_\_name\_\_ == '\_\_main\_\_':

promos = [name for name in globals()if name.endswith("\_promo")]

print(promos) #输出 ['zero\_promo', 'one\_promo', 'two\_promo']

promos = [globals()[name] for name in globals() if name.endswith("\_promo")]

print(promos[0]()) #输出 0 调用了第一个函数

#参考博客 https://www.jianshu.com/p/a9f583d8cbaa

26.hasattr()函数

'''

hasattr()函数用于判断是否包含对应的属性

语法：

hasattr(object,name)

参数：

object--对象

name--字符串，属性名

返回值：

如果对象有该属性返回True，否则返回False

'''

class People():

sex='男'

def \_\_init\_\_(self,name):

self.name=name

def peopleinfo(self):

print('欢迎%s访问'%self.name)

obj=People('zhangsan')

print(hasattr(People,'sex')) #输出 True

print('sex'in People.\_\_dict\_\_) #输出 True

print(hasattr(obj,'peopleinfo')) #输出 True

print(People.\_\_dict\_\_)

#输出 {'\_\_module\_\_': '\_\_main\_\_', 'sex': '男', '\_\_init\_\_': <function People.\_\_init\_\_ at 0x0000019931C452F0>, 'peopleinfo': <function People.peopleinfo at 0x0000019931C45378>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'People' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'People' objects>, '\_\_doc\_\_': None}

27.hash()函数

'''

描述

hash() 用于获取取一个对象（字符串或者数值等）的哈希值。

语法

hash 语法：

hash(object)

参数说明：

object -- 对象；

返回值

返回对象的哈希值。

'''

print(hash('test')) #输出 -2950866779904704330 会改变的

print(hash(1)) #数字 输出 1

print(hash(str([1,2,3]))) # 集合 输出 -6217131644886971364

print(hash(str(sorted({'1':1})))) # 字典 输出 -6233802074491902648

'''

hash() 函数可以应用于数字、字符串和对象，不能直接应用于 list、set、dictionary。

在 hash() 对对象使用时，所得的结果不仅和对象的内容有关，还和对象的 id()，也就是内存地址有关。

'''

class Test:

def \_\_init\_\_(self, i):

self.i = i

for i in range(10):

t = Test(1)

print(hash(t), id(t))

'''

-9223371889060894604 2364702099272

147793883435 2364702134960

-9223371889060894604 2364702099272

147793883435 2364702134960

-9223371889060894604 2364702099272

147793883435 2364702134960

-9223371889060894604 2364702099272

147793883435 2364702134960

-9223371889060894604 2364702099272

147793883435 2364702134960

'''

'''

hash() 函数的用途

hash() 函数的对象字符不管有多长，返回的 hash 值都是固定长度的，

也用于校验程序在传输过程中是否被第三方（木马）修改，

如果程序（字符）在传输过程中被修改hash值即发生变化，

如果没有被修改，则 hash 值和原始的 hash 值吻合，

只要验证 hash 值是否匹配即可验证程序是否带木马（病毒）。

'''

name1='正常程序代码'

name2='正常程序代码带病毒'

print(hash(name1)) # 2403189487915500087

print(hash(name2)) # -8751655075885266653

28.hex() 函数

'''

描述

hex() 函数将一个整数转换成十六进制字符串。

语法

hex 语法：

hex(x)

参数说明：

x -- 整数。

返回值

返回十六进制字符串。

'''

print(hex(12)) #输出12的八进制 0xc

print(hex(-120)) #输出-12的二进制 -0x78

print(type(hex(12))) #输出oct(12) 的类型 <class 'str'> 所以不能直接计算

print(int(hex(10),base=16)+int(hex(15),base=16)) #输出 25

#base 参数不可为空 为空默认参数为10进制 会报错 ValueError: invalid literal for int() with base 10: '0b1010'

#当然了，参数不仅可以接受十进制整数，八进制、十六进制也是可以的，只要是int型数据就合法。

print(hex(0b10010)) #输出0x12

print(hex(0o1357)) #输出0x2ef

print(hex(0x2d9)) #输出0x2d9

29id() 函数

'''

id() 函数用于获取对象的内存地址。

语法

id 语法：

id([object])

参数说明：

object -- 对象。

返回值

返回对象的内存地址。

'''

str='zhangsan'

print(id(str)) #输出 1556579882544 动态分配 id 每一次会改变

b=1

print(id(b)) #输出 1597205568

'''

id方法的返回值就是对象的内存地址。

python2中会为每个出现的对象分配内存，哪怕他们的值完全相等（注意是相等不是相同）。

如执行a=2.0，b=2.0这两个语句时会先后为2.0这个Float类型对象分配内存，

然后将a与b分别指向这两个对象。所以a与b指向的不是同一对象

python3中 值相等的变量 内存一样 如下图所示

'''

a=10.21

b=10.21

print(id(a)) #输出2036826247912

print(id(b)) #输出2036826247912

print(a is b) #输出 True

print(a == b) #输出 True

'''

id 函数 涉及到 浅拷贝和深拷贝的相关知识

深copy和浅copy

深copy新建一个对象重新分配内存地址，复制对象内容。

浅copy不重新分配内存地址，内容指向之前的内存地址。

浅copy如果对象中有引用其他的对象，如果对这个子对象进行修改，子对象的内容就会发生更改。

'''

import copy

#这里有子对象

numbers=['1','2','3',['4','5']]

#浅copy

num1=copy.copy(numbers)

#深copy

num2=copy.deepcopy(numbers)

#直接对对象内容进行修改

num1.append('6')

#这里可以看到内容地址发生了偏移，增加了偏移‘6’的地址

print('numbers:',numbers)

print('numbers memory address:',id(numbers))

print('numbers[3] memory address',id(numbers[3]))

print('num1:',num1)

print('num1 memory address:',id(num1))

print('num1[3] memory address',id(num1[3]))

num1[3].append('6')

print('numbers:',numbers)

print('num1:',num1)

print('num2',num2)

'''

输出：

numbers: ['1', '2', '3', ['4', '5']]

numbers memory address: 1556526434888

numbers memory address 1556526434952

num1: ['1', '2', '3', ['4', '5'], '6']

num1 memory address: 1556526454728

num1[3] memory address 1556526434952

numbers: ['1', '2', '3', ['4', '5', '6']]

num1: ['1', '2', '3', ['4', '5', '6'], '6']

num2 ['1', '2', '3', ['4', '5']]

'''

#参考博客 https://www.cnblogs.com/dplearning/p/5998112.html

#参考博客 https://www.cnblogs.com/JackFu/p/8014762.html

30.input() 函数

'''

Python3.x 中 input() 函数接受一个标准输入数据，返回为 string 类型。

注意：在 Python3.x 中 raw\_input() 和 input() 进行了整合，

去除了 raw\_input( )，仅保留了input( )函数，其接收任意任性输入，

将所有输入默认为字符串处理，并返回字符串类型。

函数语法

input([prompt])

参数说明：

prompt: 提示信息

'''

a=input('请输入一个数:') #输入 10

print(a) #输出 10

print(type(a)) #输出 <class 'str'>

#b=a+10 #报错 TypeError: must be str, not int

b=int(a)+10 #转换成整型

print(b) #输出 20

a=input('请输入一个字符串:') #输入 ltf1234

print(a) #输出 字符串ltf1234 可以使用字符串对应的方法

print(a.split('1')) #输出 ['ltf', '234'] split 切割字符串 直接输出列表

31.int()函数

'''

int([x[,radix]])

如果参数是字符串，那么它可能包含符号和小数点。参数radix表示转换的基数（默认是10进制）。

它可以是[2,36]范围内的值，或者0。如果是0，系统将根据字符串内容来解析。

如果提供了参数radix，但参数x并不是一个字符串，将抛出TypeError异常；

否则，参数x必须是数值（普通整数，长整数，浮点数）。通过舍去小数点来转换浮点数。

如果超出了普通整数的表示范围，一个长整数被返回。

如果没有提供参数，函数返回0。

int(x, [base])

作用：

将一个数字或base类型的字符串转换成整数。

int(x=0)

int(x, base=10)，base缺省值为10，也就是说不指定base的值时，函数将x按十进制处理。

注：

1. x 可以是数字或字符串，但是base被赋值后 x 只能是字符串

2. x 作为字符串时必须是 base 类型，也就是说 x 变成数字时必须能用 base 进制表示

'''

#1.x是数字

print(int(2.1)) #输出 2

print(int(2e3)) #输出 2000

#print(int(1000,2)) #出错，base 被赋值后函数只接收字符串

#报错 TypeError: int() can't convert non-string with explicit base

#2.x是字符串

print(int('abc12',16)) #输出 703506

#print(int('tuifyg',16)) #出错 tuifyg 超过0-9 abcdef 超出16进制

#报错 ValueError: invalid literal for int() with base 16: 'tuifyg'

#3. base 可取值范围是 2~36，囊括了所有的英文字母(不区分大小写)，

# 十六进制中F表示15，那么G将在二十进制中表示16，依此类推....Z在三十六进制中表示35

#print(int('FZ',16)) # 出错，FZ不能用十六进制表示

#报错 ValueError: invalid literal for int() with base 16: 'FZ'

print(int('FZ',36)) # 575

#4.字符串 0x 可以出现在十六进制中，视作十六进制的符号，

# 同理 0b 可以出现在二进制中，除此之外视作数字 0 和字母 x

print(int('0x10', 16)) # 16，0x是十六进制的符号

#print(int('0x10', 17)) # 出错，'0x10'中的 x 被视作英文字母 x

print(int('0x10', 36)) # 42804，36进制包含字母 x

#参考博客 https://www.cnblogs.com/guyuyuan/p/6827987.html

32isinstance() 函数

'''

isinstance() 函数来判断一个对象是否是一个已知的类型，类似 type()。

isinstance() 与 type() 区别：

type() 不会认为子类是一种父类类型，不考虑继承关系。

isinstance() 会认为子类是一种父类类型，考虑继承关系。

如果要判断两个类型是否相同推荐使用 isinstance()。

语法

以下是 isinstance() 方法的语法:

isinstance(object, classinfo)

参数

object -- 实例对象。

classinfo -- 可以是直接或间接类名、基本类型或者有它们组成的元组。

返回值

如果对象的类型与参数二的类型（classinfo）相同则返回 True，否则返回 False。。

'''

a=2

print(isinstance(a,int)) #返回 True

print(isinstance(a,str)) #返回 Fasle

print(isinstance(a,(str,int,list))) #返回 True 是元组中的一个类型 就行

class A:

pass

class B(A):

pass

print(isinstance(A(), A) ) # returns True

print(type(A()) == A ) # returns True

print(isinstance(B(), A)) # returns True

print(type(B()) == A) # returns False

33.issubclass()函数

'''

描述

issubclass() 方法用于判断参数 class 是否是类型参数 classinfo 的子类。

语法

以下是 issubclass() 方法的语法:

issubclass(class, classinfo)

参数

class -- 类。

classinfo -- 类。

返回值

如果 class 是 classinfo 的子类返回 True，否则返回 False。

'''

class A:

pass

class B(A):

pass

class C(A):

pass

print(issubclass(B, A)) # 返回 True

print(issubclass(C, A)) # 返回 True

print(issubclass(C, B)) # 返回 False

#2.class参数是classinfo的子类，并且classinfo是元组

print(issubclass(C, (A, object))) #返回 True

print(issubclass(C, (A, int, object))) #返回 True

print(issubclass(C, (int, str))) #返回 False

print(issubclass(C, (int, str, type))) #返回 False

#print(issubclass(C, (1, A)))

#报错 TypeError: issubclass() arg 2 must be a class or tuple of classes

#参考简书 https://www.jianshu.com/p/4c425bbdd773

34.iter() 函数

'''

描述

iter() 函数用来生成迭代器。

语法

以下是 iter() 方法的语法:

iter(object[, sentinel])

参数

object -- 支持迭代的集合对象。

sentinel -- 如果传递了第二个参数，则参数 object 必须是一个可调用的对象（如，函数），此时，iter 创建了一个迭代器对象，每次调用这个迭代器对象的\_\_next\_\_()方法时，都会调用 object。

打开模式

返回值

迭代器对象。

1 iter(o[, sentinel])

2 返回一个iterator对象。该函数对于第一个参数的解析依赖于第二个参数。

3 如果没有提供第二个参数，参数o必须是一个集合对象，支持遍历功能（\_\_iter\_\_()方法）或支持序列功能（\_\_getitem\_\_()方法），

4 参数为整数，从零开始。如果不支持这两种功能，将处罚TypeError异常。

5 如果提供了第二个参数，参数o必须是一个可调用对象。在这种情况下创建一个iterator对象，每次调用iterator的next()方法来无

6 参数的调用o，如果返回值等于参数sentinel，触发StopIteration异常，否则将返回该值。

'''

lst = [1,2,3,4,5,6,7]

for i in iter(lst):

print(i) #输出1,2,3,4,5,6,7

class counter:

def \_\_init\_\_(self, \_start, \_end):

self.start = \_start

self.end = \_end

def get\_next(self):

s = self.start

if(self.start < self.end):

self.start += 1

else:

raise StopIteration

return s

c = counter(1, 5)

iterator = iter(c.get\_next, 3)

print(type(iterator)) #返回 <class 'callable\_iterator'>

for i in iterator:

print(i) #输出 1 2

#参考博客 https://www.cnblogs.com/yitouniu/p/5243136.html

35.lambda()函数

'''

匿名函数lambda：是指一类无需定义标识符（函数名）的函数或子程序。

lambda 函数可以接收任意多个参数 (包括可选参数) 并且返回单个表达式的值。

要点：

1，lambda 函数不能包含命令，

2，包含的表达式不能超过一个。

说明：一定非要使用lambda函数；任何能够使用它们的地方，都可以定义一个单独的普通函数来进行替换。

我将它们用在需要封装特殊的、非重用代码上，避免令我的代码充斥着大量单行函数。

lambda匿名函数的格式：冒号前是参数，可以有多个，用逗号隔开，冒号右边的为表达式。

其实lambda返回值是一个函数的地址，也就是函数对象。

'''

def sum(x,y):

return x+y

print(sum(4,6))

f=lambda x,y:x+y

print(f(4,6))

#这俩个例子的效果是一样的，都是返回x+y

a=lambda x:x\*x

print(a(4)) #传入一个参数的lambda函数 返回x\*x

b=lambda x,y,z:x+y\*z

print(b(1,2,3)) #返回x+y\*z 即1+2\*3=7

#2.方法结合使用

from functools import reduce

foo=[2, 18, 9, 22, 17, 24, 8, 12, 27]

print(list(filter(lambda x:x%3==0,foo))) #筛选x%3==0 的元素

print(list(map(lambda x:x\*2+10,foo))) #遍历foo 每个元素乘2+10 再输出

print(reduce(lambda x,y:x+y,foo)) #返回每个元素相加的和

36.len()函数

'''

描述

Python len() 方法返回对象（字符、列表、元组等）长度或项目个数。

语法

len()方法语法：

len( s )

参数

s -- 对象。

返回值

返回对象长度。

'''

str1='ltf1234'

print(len(str1)) #输出 7

list1=[1,2,3,4,5,6,7,8]

print(len(list1)) #输出 8

for i in range(len(list1)):

print(i) #依次输出1-8

dict = {'num':777,'name':"anne"}

print(len(dict)) #输出 2

#参考博客 https://www.cnblogs.com/101718qiong/p/7542193.html

37.list()函数

'''

list()　　列表构造函数

1 list([iterable])

2 list的构造函数。参数iterable是可选的，它可以是序列，支持编译的容器对象，或iterator对象。

3 该函数创建一个元素值，顺序与参数iterable一致的列表。如果参数iterable是一个列表，将创建

4 列表的一个拷贝并返回，就像语句iterables[:]。

'''

list=[1,2,3,4,5,6,7,8,9] #构建列表

print(list) #输出 [1,2,3,4,5,6,7,8,9]

list.append(10) #列表追加 10

print(list) #输出 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

list.insert(2,18) #在列表索引为2 的位置 插入 18 其余的后移

print(list) #输出 [1, 2, 18, 3, 4, 5, 6, 7, 8, 9, 10]

print(list.count(1)) #输出 列表里1 的数量

list2=[-1,-2,-3]

list.extend(list2) #列表追加列表

print(list) #输出 [1, 2, 18, 3, 4, 5, 6, 7, 8, 9, 10, -1, -2, -3]

list.remove(1) #删除列表里的第一个1

print(list) #输出 [2, 18, 3, 4, 5, 6, 7, 8, 9, 10, -1, -2, -3]

list.sort() #列表排序

print(list) #输出 [-3, -2, -1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 18]

list.reverse() #列表反转

print(list) #输出 [18, 10, 9, 8, 7, 6, 5, 4, 3, 2, -1, -2, -3]

print(max(list)) #输出列表最大值 18

print(min(list)) #输出列表最小值 -3

list3=[1,2,3,'q','a','s']

#print(max(list3)) #报错 >' not supported between instances of 'str' and 'int'

#print(min(list3)) #报错 >' not supported between instances of 'str' and 'int'

38.locals() 函数

'''

locals() 函数会以字典类型返回当前位置的全部局部变量。

对于函数, 方法, lambda 函式, 类, 以及实现了 \_\_call\_\_ 方法的类实例, 它都返回 True。

语法

locals() 函数语法：

locals()

参数

无

返回值

返回字典类型的局部变量

1 不要修改locals()返回的字典中的内容；改变可能不会影响解析器对局部变量的使用。

2 在函数体内调用locals()，返回的是自由变量。修改自由变量不会影响解析器对变量的使用。

3 不能在类区域内返回自由变量。

'''

def test\_py(arg):

z=1

print(locals())

test\_py(6) #输出 {'z': 1, 'arg': 6}

def foo(arg, a):

x = 100

y = 'hello python!'

for i in range(10):

j = 1

k = i

print(locals())

foo(1, 2) #输出 {'k': 9, 'j': 1, 'i': 9, 'y': 'hello python!', 'x': 100, 'a': 2, 'arg': 1}

#参考博客 https://blog.csdn.net/sxingming/article/details/52061630

39.map()函数

'''

map()函数

map()是 Python 内置的高阶函数，它接收一个函数 f 和一个 list，

并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。

例如，对于list [1, 2, 3, 4, 5, 6, 7, 8, 9]

如果希望把list的每个元素都作平方，就可以用map()函数：

因此，我们只需要传入函数f(x)=x\*x，就可以利用map()函数完成这个计算：

'''

list=[1,2,3,4,5,6,7,8,9]

def f(x):

return x\*x

list1=map(f,list)

print(list1)

for i in list1:

print(i)

'''

注意：map()函数不改变原有的 list，而是返回一个新的 list。

利用map()函数，可以把一个 list 转换为另一个 list，只需要传入转换函数。

由于list包含的元素可以是任何类型，因此，map() 不仅仅可以处理只包含数值的 list，

事实上它可以处理包含任意类型的 list，只要传入的函数f可以处理这种数据类型。

任务

假设用户输入的英文名字不规范，没有按照首字母大写，后续字母小写的规则，

请利用map()函数，把一个list（包含若干不规范的英文名字）变成一个包含规范英文名字的

'''

def format\_name(s):

s1=s[0:1].upper()+s[1:].lower()

return s1

names=['adam', 'LISA', 'barT']

print (map(format\_name, names)) #python2 这样写可以直接输出列表

for i in map(format\_name,names):

print(i) #python3 得这样写才可以

40.max()函数

'''

描述

max() 方法返回给定参数的最大值，参数可以为序列。

语法

以下是 max() 方法的语法:

max( x, y, z, .... )

参数

x -- 数值表达式。

y -- 数值表达式。

z -- 数值表达式。

返回值

返回给定参数的最大值。

'''

print ("max(80, 100, 1000) : ", max(80, 100, 1000))

print ("max(-20, 100, 400) : ", max(-20, 100, 400))

print ("max(-80, -20, -10) : ", max(-80, -20, -10))

print ("max(0, 100, -400) : ", max(0, 100, -400))

'''

输出结果：

max(80, 100, 1000) : 1000

max(-20, 100, 400) : 400

max(-80, -20, -10) : -10

max(0, 100, -400) : 100

'''

#1.传入的多个参数的最大值

print(max(1,2,3,4)) #输出 4

#2.传入可迭代对象时，取其元素最大值

s='12345'

print(max(s)) #输出 5

#3.传入命名参数key，其为一个函数，用来指定取最大值的方法

s = [

{'name': 'sumcet', 'age': 18},

{'name': 'bbu', 'age': 11}

]

a = max(s, key=lambda x: x['age'])

print(a) #输出 {'name': 'sumcet', 'age': 18}

41.min()函数

'''

描述

min() 方法返回给定参数的最小值，参数可以为序列。

语法

以下是 min() 方法的语法:

min( x, y, z, .... )

参数

x -- 数值表达式。

y -- 数值表达式。

z -- 数值表达式。

返回值

返回给定参数的最小值。

'''

print ("min(80, 100, 1000) : ", min(80, 100, 1000))

print ("min(-20, 100, 400) : ", min(-20, 100, 400))

print ("min(-80, -20, -10) : ", min(-80, -20, -10))

print ("min(0, 100, -400) : ", min(0, 100, -400))

'''

输出结果

min(80, 100, 1000) : 80

min(-20, 100, 400) : -20

min(-80, -20, -10) : -80

min(0, 100, -400) : -400

'''

#1.传入的多个参数的最小值

print(min(1,2,3,4)) #输出 1

#2.传入可迭代对象时，取其元素最小值

s='12345'

print(min(s)) #输出 1

#3.传入命名参数key，其为一个函数，用来指定取最小值的方法

s = [

{'name': 'sumcet', 'age': 18},

{'name': 'bbu', 'age': 11}

]

a = min(s, key=lambda x: x['age'])

print(a) #输出 {'name': 'bbu', 'age': 11}

42.next()函数

'''

描述

next() 返回迭代器的下一个项目。

语法

next 语法：

next(iterator[, default])

参数说明：

iterator -- 可迭代对象

default -- 可选，用于设置在没有下一个元素时返回该默认值，如果不设置，又没有下一个元素则会触发 StopIteration 异常。

返回值

返回对象帮助信息。

'''

# 首先获得Iterator对象:

it = iter([1, 2, 3, 4, 5])

# 循环:

while True:

try:

# 获得下一个值:

x = next(it)

print(x)

except StopIteration:

# 遇到StopIteration就退出循环

break

a=iter('abcde')

print(next(a)) #输出 a

print(next(a)) #输出 b

print(next(a)) #输出 c

print(next(a)) #输出 d

print(next(a)) #输出 e

#print(next(a)) #报错 StopIteration

#函数可以接收一个可选的default参数，传入default参数后，

# 如果可迭代对象还有元素没有返回，则依次返回其元素值，如果所有元素已经返回，

# 则返回default指定的默认值而不抛出StopIteration 异常。

print(next(a,'e')) #这次不报错了 返回 e 即default参数

#参考博客 http://www.cnblogs.com/sesshoumaru/p/6037922.html

43.oct() 函数

'''

描述

oct() 函数将一个整数转换成八进制字符串。

语法

oct 语法：

oct(x)

参数说明：

x -- 整数。

返回值

返回八进制字符串。

'''

print(oct(12)) #输出12的八进制 0o14

print(oct(-120)) #输出-12的二进制 -0o170

print(type(oct(12))) #输出oct(12) 的类型 <class 'str'> 所以不能直接计算

print(int(oct(10),base=8)+int(oct(15),base=8)) #输出 25

#base 参数不可为空 为空默认参数为10进制 会报错 ValueError: invalid literal for int() with base 10: '0b1010'

#当然了，参数不仅可以接受十进制整数，八进制、十六进制也是可以的，只要是int型数据就合法。

print(oct(0b10010)) #输出0o22

print(oct(0o1357)) #输出0o1357

print(oct(0x2d9)) #输出0o1331

44.open() 函数

'''

python open() 函数用于打开一个文件，创建一个 file 对象，相关的方法才可以调用它进行读写。

更多文件操作可参考：Python 文件I/O。

函数语法

open(name[, mode[, buffering]])

参数说明：

name : 一个包含了你要访问的文件名称的字符串值。

mode : mode 决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

buffering : 如果 buffering 的值被设为 0，就不会有寄存。如果 buffering 的值取 1，访问文件时会寄存行。如果将 buffering 的值设为大于 1 的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

不同模式打开文件的完全列表：

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。

rb

以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。

r+

打开一个文件用于读写。文件指针将会放在文件的开头。

rb+

以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。

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，即原有内容会被删除。如果该文件不存在，创建新文件。

wb

以二进制格式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，即原有内容会被删除。如果该文件不存在，创建新文件。

w+

打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。

wb+

以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。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，文件指针将会放在文件的结尾。也就是说，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。

ab

以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，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。

a+

打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。

ab+

以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

file 对象方法

file.read([size])：size 未指定则返回整个文件，如果文件大小 >2 倍内存则有问题，f.read()读到文件尾时返回""(空字串)。

file.readline()：返回一行。

file.readlines([size]) ：返回包含size行的列表, size 未指定则返回全部行。

for line in f: print line ：通过迭代器访问。

f.write("hello\n")：如果要写入字符串以外的数据,先将他转换为字符串。

f.tell()：返回一个整数,表示当前文件指针的位置(就是到文件头的比特数)。

f.seek(偏移量,[起始位置])：用来移动文件指针。

偏移量: 单位为比特，可正可负

起始位置: 0 - 文件头, 默认值; 1 - 当前位置; 2 - 文件尾

f.close() 关闭文件

open(filename [, mode [, bufsize]])

打开一个文件，返回一个file对象。 如果文件无法打开，将处罚IOError异常。

应该使用open()来代替直接使用file类型的构造函数打开文件。

参数filename表示将要被打开的文件的路径字符串；

参数mode表示打开的模式，最常用的模式有：'r'表示读文本，'w'表示写文本文件，'a'表示在文件中追加。

Mode的默认值是'r'。

当操作的是二进制文件时，只要在模式值上添加'b'。这样提高了程序的可移植性。

可选参数bufsize定义了文件缓冲区的大小。0表示不缓冲；1表示行缓冲；任何其他正数表示使用该大小的缓冲区；

负数表示使用系统默认缓冲区大小，对于tty设备它往往是行缓冲，而对于其他文件往往完全缓冲。如果参数值被省却。

使用系统默认值。

'''

f=open('1.txt','r',encoding='utf-8')

print(f.read())

'''

输出...

ltf

zhongguo

shanxi

yuncheng

男

20

'''

#参考博客 https://www.cnblogs.com/Devilf/p/8006663.html

45.ord() 函数

'''

描述

ord() 函数是 chr() 函数（对于8位的ASCII字符串）或 unichr() 函数（对于Unicode对象）的配对函数，它以一个字符（长度为1的字符串）作为参数，返回对应的 ASCII 数值，或者 Unicode 数值，如果所给的 Unicode 字符超出了你的 Python 定义范围，则会引发一个 TypeError 的异常。

语法

以下是 ord() 方法的语法:

ord(c)

参数

c -- 字符。

返回值

返回值是对应的十进制整数。

'''

print(ord('a')) #输出97

print(ord('b')) #输出98

print(ord('c')) #输出99

print(ord(']')) #输出93

print(ord('8')) #输出56

#其实要是有一张表就好了 哈哈哈哈

46.pow()函数

'''

描述

pow() 方法返回 xy（x的y次方） 的值。

语法

以下是 math 模块 pow() 方法的语法:

import math

math.pow( x, y )

内置的 pow() 方法

pow(x, y[, z])

函数是计算x的y次方，如果z在存在，则再对结果进行取模，其结果等效于pow(x,y) %z

注意：pow() 通过内置的方法直接调用，内置方法会把参数作为整型，而 math 模块则会把参数转换为 float。

参数

x -- 数值表达式。

y -- 数值表达式。

z -- 数值表达式。

返回值

返回 xy（x的y次方） 的值。

'''

import math # 导入 math 模块

print("math.pow(100, 2) : ", math.pow(100, 2))

# 使用内置，查看输出结果区别

print("pow(100, 2) : ", pow(100, 2))

print("math.pow(100, -2) : ", math.pow(100, -2))

print("math.pow(2, 4) : ", math.pow(2, 4))

print("math.pow(3, 0) : ", math.pow(3, 0))

'''

输出结果...

math.pow(100, 2) : 10000.0

pow(100, 2) : 10000

math.pow(100, -2) : 0.0001

math.pow(2, 4) : 16.0

math.pow(3, 0) : 1.0

'''

47.property() 函数

'''

描述

property() 函数的作用是在新式类中返回属性值。

语法

以下是 property() 方法的语法:

class property([fget[, fset[, fdel[, doc]]]])

参数

fget -- 获取属性值的函数

fset -- 设置属性值的函数

fdel -- 删除属性值函数

doc -- 属性描述信息

返回值

返回新式类属性

'''

class C(object):

def \_\_init\_\_(self):

self.\_x = None

def getx(self):

return self.\_x

def setx(self, value):

self.\_x = value

def delx(self):

del self.\_x

x = property(getx, setx, delx, "I'm the 'x' property.")

class Parrot(object):

def \_\_init\_\_(self):

self.\_voltage = 100000

#装饰器写法

@property

def voltage(self):

"""Get the current voltage."""

return self.\_voltage

class D(object):

def \_\_init\_\_(self):

self.\_x = None

@property

def x(self):

"""I'm the 'x' property."""

return self.\_x

@x.setter

def x(self, value):

self.\_x = value

@x.deleter

def x(self):

del self.\_x

48.range() 函数

'''

python range() 函数可创建一个整数列表，一般用在 for 循环中。

函数语法

range(start, stop[, step])

参数说明：

start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）;

stop: 计数到 stop 结束，但不包括 stop。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5

step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

'''

for i in range(10):

print(i) #输出 从0-9

for i in range(0,11,2):

print(i) #输出 0,2,4,6,8,10

for i in range(0,-10,-3):

print(i) #输出 0,-3,-6,-9

list=[]

for i in range(5,-5,-1):

list.append(i)

print(list) #输出 [5, 4, 3, 2, 1, 0, -1, -2, -3, -4]

for i in 'ahfgohiauf':

print(i) #依次输出字符

#参考博客 https://www.cnblogs.com/101718qiong/p/7542193.html

49.reduece()函数

'''

在python3中如果使用reduce需要先导入

from functools import reduce

reduce函数，reduce函数会对参数序列中元素进行累积。

reduce函数的定义：

reduce(function, sequence [, initial] ) -> value

function参数是一个有两个参数的函数，reduce依次从sequence中取一个元素，

和上一次调用function的结果做参数再次调用function。

第一次调用function时，如果提供initial参数，

会以sequence中的第一个元素和initial作为参数调用function，

否则会以序列sequence中的前两个元素做参数调用function。

'''

from functools import reduce

lst=[1,2,3,4,5,6]

def f(x,y):

return x+y

print(reduce(f,lst))

'''

过程1+2+3+4+5+6=21

'''

print(reduce(lambda x,y:x\*y,lst))

# 运行过程为1\*2\*3\*4\*5\*6=720

#来个稍微复杂的

print(reduce(lambda x,y:x\*y+1,lst))

'''

运算步骤：1\*2+1=3

3\*3+1=10

10\*4+1=41

41\*5+1=206

206\*6+1=1237

'''

#再说说有初始化值的情况, 这个时候就不是取列表的前两项, 而是取初始值为第一个,

# 序列的第一个元素为第二个元素,开始进行lambda函数的应用计算.

print(reduce(lambda x,y:x+y,lst,5))

'''

计算步骤：5+1=6

6+2=8

8+3=11

11+4=15

15+5=20

20+6=26

'''

50.repr()函数

'''

描述

repr() 函数将对象转化为供解释器读取的形式。

语法

以下是 repr() 方法的语法:

repr(object)

参数

object -- 对象。

返回值

返回一个对象的 string 格式。

'''

s='qwerasdf'

print(s) #输出 qwerasdf

print(repr(s)) #输出 'qwerasdf'

dict={'a':1,'b':2}

print(dict) #输出 {'a': 1, 'b': 2}

print(repr(dict)) #输出 {'a': 1, 'b': 2} 没改变么

print(repr([0,1,2,3,4])) #输出 [0, 1, 2, 3, 4]

print(repr('hello')) #输出 'hello'

print(str(1.0/7.0)) #输出 0.14285714285714285

print(repr(1.0/7.0)) #输出 0.14285714285714285

51.reversed() 函数

'''

描述

reversed 函数返回一个反转的迭代器。

语法

以下是 reversed 的语法:

reversed(seq)

参数

seq -- 要转换的序列，可以是 tuple, string, list 或 range。

返回值

返回一个反转的迭代器。

1 reversed(seq)

2 返回一个逆序的iterator对象。参数seq必须是一个包含\_\_reversed\_\_()方法的对象或支持序列操作(\_\_len\_\_()和\_\_getitem\_\_())

3 该函数是2.4中新增的

'''

str='wasdqwer'

print(list(reversed(str))) #输出 ['r', 'e', 'w', 'q', 'd', 's', 'a', 'w']

tuple=('r', 'e', 'w', 'q', 'd', 's', 'a', 'w')

print(list(reversed(tuple))) #输出 ['w', 'a', 's', 'd', 'q', 'w', 'e', 'r']

seqRange = range(5, 9)

print(list(reversed(seqRange))) #输出 [8, 7, 6, 5]

seqList = [1, 2, 4, 3, 5]

print(list(reversed(seqList))) #输出 [5, 3, 4, 2, 1]

a=[1,2,3,4,5,6]

b=reversed(a)

print(b) #输出 <list\_reverseiterator object at 0x0000023E2A448748> 显示为一个迭代器

print(list(b)) #输出 [6, 5, 4, 3, 2, 1]

print(list(b)) #输出 []

#由此可知：reversed（）返回的是一个迭代器对象，只能进行一次循环遍历。显示一次所包含的值！

52.round()函数

'''

描述

round() 方法返回浮点数x的四舍五入值。

语法

以下是 round() 方法的语法:

round( x [, n] )

参数

x -- 数字表达式。

n -- 表示从小数点位数，其中 x 需要四舍五入，默认值为 0。

返回值

返回浮点数x的四舍五入值。

'''

print ("round(70.23456) : ", round(70.23456))

print ("round(56.659,1) : ", round(56.659,1))

print ("round(80.264, 2) : ", round(80.264, 2))

print ("round(100.000056, 3) : ", round(100.000056, 3))

print ("round(-100.000056, 3) : ", round(-100.000056, 3))

'''

输出结果...

round(70.23456) : 70

round(56.659,1) : 56.7

round(80.264, 2) : 80.26

round(100.000056, 3) : 100.0

round(-100.000056, 3) : -100.0

'''

53.setattr()函数

'''

描述：

setattr函数，用于设置属性值，该属性必须存在

语法：

setattr(object,name,value)

参数：

object--对象

name--字符串，对象属性

value--属性值

返回值：

无

'''

class People():

sex='男'

def \_\_init\_\_(self,name):

self.name=name

def peopleinfo(self):

print('欢迎%s访问'%self.name)

obj=People('zhangsan')

setattr(People,'x',123)

print(People.x) #等同于 Peopel.x=123

setattr(obj,'age',18)

print(obj.\_\_dict\_\_) #输出 {'name': 'zhangsan', 'age': 18}

print(People.\_\_dict\_\_)

#输出

#{'\_\_module\_\_': '\_\_main\_\_', 'sex': '男', '\_\_init\_\_': <function People.\_\_init\_\_ at 0x00000259A92752F0>, 'peopleinfo': <function People.peopleinfo at 0x00000259A9275378>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'People' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'People' objects>, '\_\_doc\_\_': None, 'x': 123}

54.set() 函数

'''

描述

set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等。

语法

set 语法：

class set([iterable])

参数说明：

iterable -- 可迭代对象对象；

返回值

返回新的集合对象。

'''

a=set('www.baidu.com')

b=set('www.gogle.com') #重复的元素被删除 元素唯一 可以用来去重

print(a) #输出 {'u', '.', 'm', 'c', 'w', 'd', 'i', 'a', 'o', 'b'}

print(b) #输出 {'.', 'm', 'e', 'c', 'w', 'o', 'l', 'g'}

print(a&b) #交集 {'m', 'c', 'w', '.', 'o'}

print(a|b) #并集 {'m', 'c', 'i', 'w', 'b', 'd', 'u', 'g', 'e', 'a', '.', 'o', 'l'}

print(a-b) #差集 {'i', 'b', 'd', 'u', 'a'}

#1.比较

se = {11, 22, 33}

be = {22, 55}

temp1 = se.difference(be) #找到se中存在，be中不存在的集合，返回新值

print(temp1) #{33, 11}

print(se) #{33, 11, 22}

temp2 = se.difference\_update(be) #找到se中存在，be中不存在的集合，覆盖掉se

print(temp2) #None

print(se) #{33, 11},

#2.删除

se = {11, 22, 33}

se.discard(11)

se.discard(44) # 移除不存的元素不会报错

print(se)

se = {11, 22, 33}

se.remove(11)

#se.remove(44) # 移除不存的元素会报错

print(se)

se = {11, 22, 33} # 移除末尾元素并把移除的元素赋给新值

temp = se.pop()

print(temp) # 33

print(se) # {11, 22}

#3.取交集

se = {11, 22, 33}

be = {22, 55}

temp1 = se.intersection(be) #取交集，赋给新值

print(temp1) # 22

print(se) # {11, 22, 33}

temp2 = se.intersection\_update(be) #取交集并更新自己

print(temp2) # None

print(se) # 22

#4.判断

se = {11, 22, 33}

be = {22}

print(se.isdisjoint(be)) #False，判断是否不存在交集（有交集False，无交集True）

print(se.issubset(be)) #False，判断se是否是be的子集合

print(se.issuperset(be)) #True，判断se是否是be的父集合

#5.合并

se = {11, 22, 33}

be = {22}

temp1 = se.symmetric\_difference(be) # 合并不同项，并赋新值

print(temp1) #{33, 11}

print(se) #{33, 11, 22}

temp2 = se.symmetric\_difference\_update(be) # 合并不同项，并更新自己

print(temp2) #None

print(se) #{33, 11}

#6.取并集

se = {11, 22, 33}

be = {22,44,55}

temp=se.union(be) #取并集，并赋新值

print(se) #{33, 11, 22}

print(temp) #{33, 22, 55, 11, 44}

#7.更新

se = {11, 22, 33}

be = {22,44,55}

se.update(be) # 把se和be合并，得出的值覆盖se

print(se)

se.update([66, 77]) # 可增加迭代项

print(se)

#8.集合的转换

se = set(range(4))

li = list(se)

tu = tuple(se)

st = str(se)

print(li,type(li)) #输出 [0, 1, 2, 3] <class 'list'>

print(tu,type(tu)) #输出 [0, 1, 2, 3] <class 'tuple'>

print(st,type(st)) #输出 [0, 1, 2, 3] <class 'str'>

#参考博客 https://www.cnblogs.com/whatisfantasy/p/5956775.html

55.slice() 函数

'''

描述

slice() 函数实现切片对象，主要用在切片操作函数里的参数传递。

语法

slice 语法：

class slice(stop)

class slice(start, stop[, step])

参数说明：

start -- 起始位置

stop -- 结束位置

step -- 间距

返回值

返回一个切片对象。

实例

'''

myslice=slice(5) #设置一个 截取五个元素的切片

print(myslice) #输出 slice(None, 5, None)

arr=list(range(10))

print(arr) #输出 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(arr[myslice]) #输出 [0, 1, 2, 3, 4]

print(arr[3:6]) #输出 [3, 4, 5]

56.sorted() 函数

'''

描述

sorted() 函数对所有可迭代的对象进行排序操作。

sort 与 sorted 区别：

sort 是应用在 list 上的方法，sorted 可以对所有可迭代的对象进行排序操作。

list 的 sort 方法返回的是对已经存在的列表进行操作，而内建函数 sorted 方法返回的是一个新的 list，而不是在原来的基础上进行的操作。

语法

sorted 语法：

sorted(iterable, key=None, reverse=False)

参数说明：

iterable -- 可迭代对象。

key -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。

reverse -- 排序规则，reverse = True 降序 ， reverse = False 升序（默认）。

返回值

返回重新排序的列表。

'''

print(sorted([2,3,4,1,5,6])) #输出 [1, 2, 3, 4, 5, 6]

#另一个区别在于list.sort() 方法只为 list 定义。而 sorted() 函数可以接收任何的 iterable。

print(sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})) #输出 [1, 2, 3, 4, 5]

#利用key进行倒序排序

example\_list = [5, 0, 6, 1, 2, 7, 3, 4]

result\_list = sorted(example\_list, key=lambda x: x\*-1)

print(result\_list) #输出 [7, 6, 5, 4, 3, 2, 1, 0]

#要进行反向排序，也通过传入第三个参数 reverse=True：

example\_list = [5, 0, 6, 1, 2, 7, 3, 4]

result\_list=sorted(example\_list, reverse=True)

print(result\_list) #输出 [7, 6, 5, 4, 3, 2, 1, 0]

#sorted 的应用，也可以通过 key 的值来进行数组/字典的排序，比如

array = [{"age":20,"name":"a"},{"age":25,"name":"b"},{"age":10,"name":"c"}]

array = sorted(array,key=lambda x:x["age"])

print(array) #输出 [{'age': 10, 'name': 'c'}, {'age': 20, 'name': 'a'}, {'age': 25, 'name': 'b'}]

57.staticmethod()函数

'''

python staticmethod 返回函数的静态方法。

该方法不强制要求传递参数，如下声明一个静态方法：

class C(object):

@staticmethod

def f(arg1, arg2, ...):

...

以上实例声明了静态方法 f，类可以不用实例化就可以调用该方法 C.f()，当然也可以实例化后调用 C().f()。

函数语法

staticmethod(function)

参数说明：

无

'''

class C(object):

@staticmethod

def f():

print('hello world')

C.f() # 静态方法无需实例化

cobj = C()

cobj.f() # 也可以实例化后调用

class A(object):

def foo(self, x):

print("executing foo(%s,%s)" % (self, x))

print('self:', self)

@classmethod

def class\_foo(cls, x):

print("executing class\_foo(%s,%s)" % (cls, x))

print('cls:', cls)

@staticmethod

def static\_foo(x):

print("executing static\_foo(%s)" % x)

a = A()

print(a.foo) #输出 <bound method A.foo of <\_\_main\_\_.A object at 0x000001B5B2A51D30>>

print(a.class\_foo) #输出 <bound method A.class\_foo of <class '\_\_main\_\_.A'>>

print(a.static\_foo) #输出 <function A.static\_foo at 0x000001B5B2A55598>

#参考博客 https://www.cnblogs.com/elie/p/5876210.html

58.str() 函数

'''

描述

str() 函数将对象转化为适于人阅读的形式。

语法

以下是 str() 方法的语法:

class str(object='')

参数

object -- 对象。

返回值

返回一个对象的string格式。

'''

print(str(1)) #输出 字符串1

print(type(str(1))) #输出 <class 'str'>

print(str(b'\xe5\xbc\xa0\xe4\xb8\x89',encoding='utf-8')) #输出张三

dict={'zhangsan':'zhang1234','lisi':'li1234'}

print(type(dict)) #输出 <class 'dict'>

a=str(dict)

print(str(dict)) #输出 字符串 {'zhangsan': 'zhang1234', 'lisi': 'li1234'}

print(type(a)) #输出 <class 'str'>

59.sum()函数

'''

描述

sum() 方法对系列进行求和计算。

语法

以下是 sum() 方法的语法:

sum(iterable[, start])

参数

iterable -- 可迭代对象，如：列表、元组、集合。

start -- 指定相加的参数，如果没有设置这个值，默认为0。

返回值

返回计算结果

'''

print(sum([0,1,2])) # 列表总和 3

print(sum((2,3,4),1)) # 元组计算总和后再加 1

print(sum([2,3,4,5,6],8)) # 列表计算总和后再加 2

a = list(range(1,11))

b = list(range(1,10))

c = sum([item for item in a if item in b])

print(c) #输出 45

60.vars() 函数

'''

描述

vars() 函数返回对象object的属性和属性值的字典对象。

语法

vars() 函数语法：

vars([object])

参数

object -- 对象

返回值

返回对象object的属性和属性值的字典对象，如果没有参数，就打印当前调用位置的属性和属性值 类似 locals()。

'''

print(vars())

#输出 {'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': '\n描述\nvars() 函数返回对象object的属性和属性值的字典对象。\n语法\nvars() 函数语法：\nvars([object])\n参数\nob

class A:

a=1

\_\_dict\_\_ = 'ltf'

print(vars(A))

#输出 {'\_\_module\_\_': '\_\_main\_\_', 'a': 1, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'A' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'A' objects>, '\_\_doc\_\_': None}

a=A()

print(vars(a))

#输出 ltf

print(a.\_\_dict\_\_)

#输出 ltf

61.zip()函数

#zip函数接受任意多个可迭代对象作为参数,将对象中对应的元素打包成一个tuple,然后返回一个可迭代的zip对象.

#这个可迭代对象可以使用循环的方式列出其元素

#若多个可迭代对象的长度不一致,则所返回的列表与长度最短的可迭代对象相同.

#1.用列表生成zip对象

x=[1,2,3]

y=[4,5,6]

z=[7,8,9]

h=['a','b','c','d']

zip1=zip(x,y,z)

print(zip1)

for i in zip1:

print(i)

zip2=zip(x,y,h)

for i in zip2:

print(i)

zip3=zip(h)

for i in zip3:

print(i)

zip4=zip(\*h\*3)

for i in zip4:

print(i) #这是干啥哟。。

print('==\*=='\*10)

#2.二维矩阵变换

l1=[[1,2,3],[4,5,6],[7,8,9]]

print(l1)

print([[j[i] for j in l1] for i in range(len(l1[0])) ])

zip5=zip(\*l1)

for i in zip5:

print(i)