### 欢迎大家关注我的公众号『Python3X』，扫描下方二维码( 或微信搜索：python3xxx)。即可关注。

主要分享Python类技术干货，包括不限于Python技术、Web开发、爬虫、数据分析等，当然还有不定期的工具干货分享。

Image text

Image text

# 目录索引：

[001、Python中的可变对象与不可变对象](#001python中的可变对象与不可变对象)

[002、类的定义和装饰器@classmethod与@staticmethod](#002类的定义和装饰器classmethod与staticmethod)

[003、删除列表中的重复元素](#003删除列表中的重复元素)

[004、Python中的lambda表达式](#004python中的lambda表达式)

[005、Python中的可变参数](#005python中的可变参数)

[006、Python中的super()用法](#006python中的super用法)

[007、Python中的深浅拷贝](#007python中的深浅拷贝)

[008、Python中文件的读取](#008python中文件的读取)

[009、is 与 == 的区别](#009is-与--的区别)

[010、xs和.format的主要区别是什么](#010s和format的主要区别是什么)

[011、列表和元组有哪些区别](#011列表和元组有哪些区别)

[012、解释一下Python中的三元运算](#012解释一下python中的三元运算)

[013、join() 和 split() 函数](#013join-和-split-函数)

[014、一行代码求 1+2+3..+100的和](#014一行代码求-1--2--3----100的和)

[015、怎么样合并两个字典](#015怎么样合并两个字典)

[016、Python中的数据类型有哪些](#016python中的数据类型有哪些)

[017、读写文件时，open与with...open .. as的区别](#017读写文件时open与with-open--as的区别)

[018、filter方法求出列表所有奇数并构造新列表](#018filter方法求出列表所有奇数并构造新列表)

[019、列表推导式求列表所有奇数并构造新列表](#019列表推导式求列表所有奇数并构造新列表)

[020、抛出自定义异常](#020抛出自定义异常)

[021、Python单引号、双引号、三引号的区别？](#021python单引号双引号三引号的区别)

[022、如何去除字符串中多个空格并保留一个](#022如何去除字符串中多个空格并保留一个)

[023、在字典中操作中pop 和 del 有什么区别](#023在字典中操作中pop-和-del-有什么区别)

[024、[[1,2],[3,4],[5,6]] 用一行代码展开该列表，得出[1,2,3,4,5,6]](#024123456-用一行代码展开该列表得出123456)

[025、如何打乱列表的顺序？](#025如何打乱列表的顺序)

[026、什么是PEP8](#026什么是pep8)

[027、如何将两个列表转换成字典，如['a', 'b', 'c'], [1, 2, 3] 转换成{'a': 1, 'b' : 2, 'c': 3}](#027如何将两个列表转换成字典如a-b-c-1-2-3-转换成a-1-b--2-c-3)

[028、Python中十进制、十六进制、字符串、字节串之间的转换](#028python中十进制十六进制字符串字节串之间的转换)

[029、.pyc 与 .py文件有什么区别](#029pyc-与-py文件有什么区别)

[030、Python中pass、continue、break、return的区别](#030python中passcontinuebreakreturn的区别)

[031、python中的异常处理模块](#031python中的异常处理模块)

[032、python中有哪些标准异常类](#032python中有哪些标准异常类)

[033、列出Python中的几个魔法方法，并简述用途](#033列出python中的几个魔法方法并简述用途)

[034、sort和sorted的区别](#034sort和sorted的区别)

[035、s="info:xiaoZhang 33 shandong",用正则切分字符串输出['info', 'xiaoZhang', '33', 'shandong']](#035sinfoxiaozhang-33-shandong用正则切分字符串输出info-xiaozhang-33-shandong)

[036、怎么样在函数内修改全局变量](#036怎么样在函数内修改全局变量)

[037、简单描述一下你常用的五个标准库](#037简单描述一下你常用的五个标准库)

[038、简述一下Python中的GIL](#038简述一下python中的gil)

--

## 001、Python中的可变对象与不可变对象

对象分为 **可变对象** 和 **不可变对象**。 可通过函数 id() 查看对象的内存地址是否改变

* 不可变对象：改变某个变量时候，由于其所指的值不能被改变，相当于把原来的值复制一份后再改变，这会开辟一个新的地址，变量再指向这个新的地址。 如：String、Tuple、Number。他们本身的值是不可以被改变，修改的时候，会复制一个新的对象，并开辟一份新的内存空间，变量再去指向新的值。
* a = 1  
  print(id(a)) # 4543532032  
  a = 1 + 2  
  print(id(a)) # 4543532096  
  # 证明字符串是可变对象
* 可变对象：意味着修改该数据对象，不会在内存中新创建另一个内存空间来存放新数据对象，如：List、Dict、Set。对其进行修改时，并不会像可变对象那样重新复制一份。而是在原有的基础上进行修改。
* a = [1, 2]  
  print(id(a)) # 4362993544  
  a.append(3)  
  print(id(a)) # 4362993544  
  a[1] = 5  
  print(id(a)) # 4362993544  
  # 列表是不可变对象

## 002、类的定义和装饰器@classmethod与@staticmethod

参考此文：https://mp.weixin.qq.com/s/t1T\_K5wbovOAIdzgKjIwhg

## 003、删除列表中的重复元素

1、通过set方法

a = [1, 2, 3, 1, 1, 1, 7, 9, 5]  
print(list(set(a)))

2、通过fromkeys方法

a = [1, 2, 3, 1, 1, 1, 7, 9, 5]  
b = {}  
# fromnkeys 创建一个新的字典，已a中的元素作为字典的键  
b = b.fromkeys(a)  
print(b) # {1: None, 2: None, 3: None, 7: None, 9: None, 5: None}  
c = list(b.keys())  
print(c) # [1, 2, 3, 7, 9, 5]

3、逻辑判断

a = [1, 2, 3, 1, 1, 1, 7, 9, 5]  
b = []  
  
for i in a:  
 if i not in b:  
 b.append(i)  
  
print(b) # [1, 2, 3, 7, 9, 5]

## 004、Python中的lambda表达式

lambda又称为匿名函数，可以不再去定义函数，使代码更加简洁。

def m(x):  
 return x \*\* 2  
print(m(2))  
# 等同于函数m()  
m1 = lambda x: x \*\* 2  
print(m1(2))

更多用法，请参考此文：https://mp.weixin.qq.com/s/CsBpTh0cDyyOT0BZ5cUHwg

## 005、Python中的可变参数

可变参数的形式有

* *args : 元组类型参数 ```python def fun\_arr(name,* args): print(name) print(args)

if **name** == '**main**': fun\_arr('tom') # 输出 tom 、() # tom # (20, 'man', '篮球', 'pythoner', '17766666666') fun\_arr('tom', 20, 'man', '篮球', 'pythoner', '17766666666') ```

* **kwargs ：字典类型参数 ```python def fun\_dict(**kwargs): for k, v in kwargs.items(): print(k, v) # 也可通过get的方式取参数的值 # my name is tom i'am 18 years old, my phone number is 17766666666 print("my name is %s i'am %s years old, my phone number is %s" % (kwargs.get('name'), kwargs.get('age'), kwargs.get('mobile')))

if **name** == '**main**': fun\_dict(name='tom', age=18, mobile='17766666666') ```

可变参数允许多个任意参数，也允许不传参数。

两个可变参数同时存在是，一定要将 \*参数 放在 \*\*参数 之前！！！否则编译错误。

通常情况下，都是将可变参数写在最后面。

如果写在前面则会将后面的参数也会收集到列表中。如果将可变参数写在前面，则需要指定后面的参数名

## 006、Python中的super()用法

super()是Python中内置的一个函数，用于调用父类的方法。

class A():  
 def eat(self):  
 print('i eat food')  
  
class B(A):  
 def run(self):  
 print('i can run')  
  
b = B()  
# 调用父类的eat  
b.eat() # i eat food

**如果一个子类继承了两个父类，并且父类中的方法名相同。那究竟会调用哪个类呢？**

这个问题可以通过类的内置属性 **mro** 去查看调用顺序。这个方法主要是在多继承时判断方法、属性的调用路径。 ```python class A: num = 1 def method(self): print('A ..method')

class B: num = 2 def method(self): print('B .. method')

class C(A, B): num = 3 pass

if **name** == '**main**': c = C() print(C.**mro**) # (, , , ) print(c.num) # 3 c.method() # A ..method ``` 如上，C.**mro** 打印结果可以执行顺序为 C -> A - >B

因此print(c.num)。因为c中有num属性，所以会打印3.

执行c.method时 。因为c中没有method属性，会去A中查找，则输出 A..method。

## 007、Python中的深浅拷贝

拷贝就是对变量的复制， > 浅拷贝：

拷贝对象并给其分配新的内存。但是只会拷贝父对象

# 定义一个变量a  
a = ['a', 1, 'd', ['z', 'x', 'c']]  
# b 浅拷贝了a  
b = a.copy()  
  
print(a) # ['a', 1, 'd', ['z', 'x', 'c']]  
print(b) # ['a', 1, 'd', ['z', 'x', 'c']]  
# 打印各自的内存地址，发现各不相同。证明了浅拷贝会分配一个新的内存地址  
print(id(a)) # 2767408  
print(id(b)) # 121575744

但是如果修改a中的元素，会怎么样呢？

# 修改a中的第一个元素为'A'  
a[0] = 'A'  
print(a) # ['A', 1, 'd', ['z', 'x', 'c']]  
print(b) # ['a', 1, 'd', ['z', 'x', 'c']] 发现b并没有受影响  
  
# 修改第四个元素中的第一个元素，将z 修改成 Q  
a[3][1] = 'Q'  
print(a) # ['A', 1, 'd', ['z', 'Q', 'c']]  
print(b) # ['A', 1, 'd', ['z', 'Q', 'c']] 发现b也被修改了。why？  
  
print(id(a[3])) # 2766208  
print(id(b[3])) # 2766208

如上修改a中的第一个元素。b不会被修改。但是修改 第四个元素中的第一个元素。b却被影响了。这就是前面说的浅拷贝只会拷贝第一层数据。

第二层['z', 'Q', 'c']并没有拷贝成功。

这个时候b中的['z', 'Q', 'c'] 的内存地址和a中的['z', 'Q', 'c']内存地址是一样的。 (浅拷贝只是把这个第二层的引用拷贝过来了。)。所以a中第二层的数据发生变化，b中的数据也会跟着变化。

深拷贝：

对对象的完全拷贝，不管你有多少层，数据都不会共享。 深拷贝需要引入copy模块。import copy

还是拿上面的数据我们进行演示。

import copy  
  
a = ['a', 1, 'd', ['z', 'x', 'c']]  
b = copy.deepcopy(a)  
  
print(a) # ['a', 1, 'd', ['z', 'x', 'c']]  
print(a) # ['a', 1, 'd', ['z', 'x', 'c']]  
# 变量的内存空间地址  
print(id(a)) # 120265424  
print(id(b)) # 15809072  
print(a[-1])  
# 最后一个元素(['z', 'x', 'c'])的内存地址  
print(id(a[-1])) # 16171936  
print(id(b[-1])) # 120265344  
  
# 修改  
a[0] = 'A'  
a[3][0] ='Q'  
print(a) # ['A', 1, 'd', ['Q', 'x', 'c']]  
# 深拷贝中第二层数据也是独立的,不会被修改  
print(b) # ['a', 1, 'd', ['z', 'x', 'c']]

## 008、Python中文件的读取

读取文件的方式有三种

* read()
* readline()
* readlines()

以上都可以通过传入一个int类型的整数来限制读取范围。

创建文件的open函数用法：

file = open(file\_name, mode='r'，encoding=)

**file\_name**：文件路径

**mode**：打开方式：默认'r',

**encoding**: 指定打开的编码

常用的打开方式：

* **r**：以只读的方式打开文件，文件必须存在
* **r+** ：以读写的方式打开文件，文件必须存在
* **w**：以写的方式打开文件，文件若存在，则清空内容，从头写入。若不存在自动创建
* **w+**：已读写的方式打开文件，文件若存在，则清空内容，从头写入。若不存在自动创建
* **a**：已写的方式打开文件，如果存在，则后面追加写入，如果不存在，则创建
* **a+**：已读写的方式打开文件，如果存在，则后面追加写入，如果不存在，则创建

read()

一次性读取整个文件，以字符串的形式返回，如果文件过大，会占用较大的内存空间。

try:  
 # 只读模式，utf-8编码打开  
 file = open('古诗', mode='r', encoding='utf-8')  
 # 床前明月光，  
 print(file.read(6)) # 读取前6个字符。如果不填 输入全部  
except (FileNotFoundError, IOError) as e:  
 print('文件异常')  
finally:  
 file.close()

上述代码是一种相对比较low的写法，因为每次读取文件都要关闭文件。这样做是为了操作文件占用的操作系统资源。所以一般读写文件都不这样写，而是通过with open..as..的形式去完成。

with open('text', 'r', encoding='utf-8') as file:  
 """  
 床前明月光，  
 疑是地上霜。  
 举头望明月，  
 低头思故乡。  
 """  
 print(file.read())

readline()

按行读取文件，读取大文件的时候因为一次只读一行数据，所以不会占用较大内存。

with open('text', 'r', encoding='utf-8') as file:  
 while 1:  
 line = file.readline()  
 if not line:  
 break  
 # readline() 每读取一行内容会有个换行符。  
 # print(line.strip())  
 print(line.replace('\n', ''))

readlines()

一次读取所有文件内容，返回列表，列表中的每个元素 代表每一行的内容。

with open('古诗', 'r', encoding='utf-8') as file:  
 lines = file.readlines()  
 # ['床前明月光，\n', '疑是地上霜。\n', '举头望明月，\n', '低头思故乡。']  
 print(lines)  
 for i in lines:  
 print(i.strip())

## 009、is 与 == 的区别

* is：判断两个变量的引用是否相等，值相同，引用不一定相同。
* ==：判断两个变量的值是否相等，如果引用相同，则值一定相同

对于整数来说，如果是在[-5，256]之间的整数来说，他们的引用也是相同的，如果不在这个范围则会重新分配内存，引用自然就不同了。

注：在Pycharm中，对Python解释器进行了优化，如果超过这个范围 a is b仍然为True。 下方示例代码，为终端运行

>>> a = -5  
>>> b = -5  
>>> print(a is b)  
True  
>>> a = -6  
>>> b = -6  
>>> print(a is b)  
False  
>>> a = 256  
>>> b = 256  
>>> print(a is b)  
True  
>>> a = 266  
>>> b = 266  
>>> print(a is b)  
False  
>>> a = 6.666  
>>> b = 6.666  
>>> print(a is b)  
False

## 010、%s和.format的主要区别是什么

%s用法，代表一个参数，写多少个%s，后面的tuple就要有多少个元素，切按照顺序一一对应

# my name is tom i am 19 years old  
print('my name is %s i am %d years old' % ('tom', 19))

format用法：format用{}的形式表示，对顺序没有严格要求

# my name is tom i am 19 years old  
print('my name is {} i am {} years old'.format('tom', 19))  
# my name is tom i am 19 years old  
print('my name is {1} i am {0} years old'.format(19, 'tom'))  
  
# hello, world, hello  
print('{0}, {1}, {0}'.format('hello', 'world'))

## 011、列表和元组有哪些区别

二者的主要区别是列表是可变的，而元组是不可变的。举个例子，如下所示：

arr = ['a', 'b', 'c']  
tuple\_arr = ('a', 'b', 'c')  
  
arr[1] = 'z'  
print(arr) # ['a', 'z', 'c']  
tuple\_arr[1] = 'z' # 报错

当执行最后一行时报错，TypeError: 'tuple' object does not support item assignment

## 012、解释一下Python中的三元运算

python没有其他语言的三元表达式，在python中只有类似的替代办法。

def fun(a: int, b: int):  
 return 'a' if a > b else 'b'  
  
print(fun(1, 5)) # b  
print(fun(5, 1)) # a

如上函数fun意思为，当a > b 返回a，否则返回b

## 013、join() 和 split() 函数

**join()** 将指定字符添加入字符串中

print(','.join('abcd')) # a,b,c,d  
print(','.join(['a', 'b', 'c', 'd'])) # a,b,c,d

**split** 将字符串分割成列表

temp = 'a,b,c,d'  
temp\_2 = 'a-b-c-d'  
print(temp.split(',')) # ['a', 'b', 'c', 'd']  
print(temp\_2.split('-')) # ['a', 'b', 'c', 'd']

## 014、一行代码求 1 + 2 + 3 + .. + 100的和

temp = sum(range(1, 101))  
print(temp) # 5050

## 015、怎么样合并两个字典

dict\_1 = {'a': 1, 'b': 2}  
dict\_2 = {'c': 3, 'd': 5}  
  
"""  
方式一  
"""  
print(dict(dict\_1, \*\*dict\_2)) # {'a': 1, 'b': 2, 'c': 3, 'd': 5}  
  
"""  
方式二  
"""  
dict\_1.update(dict\_2)  
print(dict\_1)  
  
  
"""  
方式三  
"""  
for k, v in dict\_1.items():  
 dict\_2[k] = v  
  
print(dict\_2)

## 016、Python中的数据类型有哪些

* 数字 (Number)
* 字符串 (str)
* 列表 (list)
* 字典 (dict)
* 元组 (tuple)
* 布尔 (Boolean)
* 集合 (set)

## 017、读写文件时，open与with open .. as的区别

使用open时，处理结束时，要释放资源。即f.close();在使用open打开文件时， open('text.txt') 如果文件不存在，则会抛出异常。这样读取文件所占用的资源就无法及时被释放。 因此通常配合try..except..finally 使用。 而with open 则不再需要我们去执行close

## 018、filter方法求出列表所有奇数并构造新列表

filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。该接收两个参数，第一个为函数，第二个为序列，序列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 的元素放到新列表

a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
def func(a):  
 return a % 2 == 1  
  
new\_list = filter(func, a)  
new\_list = [i for i in new\_list]  
print(new\_list) # [1, 3, 5, 7, 9]

## 019、列表推导式求列表所有奇数并构造新列表

a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
new\_list = [i for i in a if i % 2 == 1]  
print(new\_list)

## 020、抛出自定义异常

当程序出现错误，python会自动引发异常，也可以通过raise显示地引发.

def fn():  
 for i in range(10):  
 if i > 5:  
 raise Exception('数字大小不能超过5')  
  
fn()

## 021、Python单引号、双引号、三引号的区别？

单引号、双引号功能一样，都可以表示字符串，也都可以起到转义的功能。在转义这一点上，三个引号也有这样的功能。 区别在于三引号可以显示多行。

"""hello  
   
world  
"""

## 022、如何去除字符串中多个空格并保留一个

text = 'aaaa cccc'  
print(' '.join(text.split())) # aaaa cccc

## 023、在字典中操作中pop 和 del 有什么区别

#### 1. 通过pop进行删除

temp = {'a': 1, 'b': 2, 'c': 3}  
  
# 删除a元素 并将对应的值赋值给v  
v = temp.pop('a')  
print(v) # 1  
print(temp) # {'b': 2, 'c': 3}  
  
# 如果元素不存在，可以设置值返回值。否则会报错  
v = temp.pop('d', 'not exist')  
print(v) # not exist  
  
v = temp.pop('d') # KeyError: 'd'  
print(v)

#### 2.通过del进行删除

del 不像pop那样有返回值。

temp = {'a': 1, 'b': 2, 'c': 3}  
``  
  
del temp['a']  
print(temp) # {'b': 2, 'c': 3}  
  
del temp['d'] # KeyError: 'd'

## 024、[[1,2],[3,4],[5,6]] 用一行代码展开该列表，得出[1,2,3,4,5,6]

a = [[1, 2], [3, 4], [5, 6]]  
b = [j for i in a for j in i]  
print(b) # [1, 2, 3, 4, 5, 6]

## 025、如何打乱列表的顺序？

a = ['a', 'b', 'c', 'd', 'e']  
import random  
random.shuffle(a) # 不会生成新的对象，改变的是a  
print(a) # ['e', 'a', 'b', 'c', 'd']

## 026、什么是PEP8

PEP8是Python官方推出的编码约定，主要是为了保证Python编码风格的统一。提高代码的可读性 比如： - 缩进。4个空格的缩进（编辑器都可以完成此功能），不使用Tap，更不能混合使用Tap和空格。

* 每行最大长度79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。
* 类和top-level函数定义之间空两行；类中的方法定义之间空一行；函数内逻辑无关段落之间空一行；其他地方尽量不要再空行。
* 模块导入的顺序：按标准、三方和自己编写顺序依次导入，之间空一行。
* 不要在一句import中多个库，比如import os, sys不推荐
* 避免不必要的空格
* 注释必须要有
* 函数命名要遵循规范
* 尽可能使用‘is’‘is not’取代‘==’，比如if x is not None 要优于if x。
* 使用基于类的异常，每个模块或包都有自己的异常类，此异常类继承自Exception。
* 异常中try的代码尽可能少。

## 027、如何将两个列表转换成字典，如['a', 'b', 'c'], [1, 2, 3] 转换成{'a': 1, 'b' : 2, 'c': 3}

a1 = ['a', 'b', 'c']  
a2 = [1, 2, 3]  
print(dict(zip(a1, a2)))

## 028、Python中十进制、十六进制、字符串、字节串之间的转换

参考阅读：https://mp.weixin.qq.com/s/9Aj6VhI503Zdpmr8uRw3sg

## 029、.pyc 与 .py文件有什么区别

在Python程序中，是把原始程序代码放在.py文件里，而Python会在执行.py文件的时候。将.py形式的程序编译成中间式文件（byte-compiled）的.pyc文件，这么做的目的就是为了加快下次执行文件的速度。

所以，在我们运行python文件的时候，就会自动首先查看是否具有.pyc文件，如果有的话，而且.py文件的修改时间和.pyc的修改时间一样，就会读取.pyc文件，否则，Python就会读原来的.py文件。

其实并不是所有的.py文件在与运行的时候都会产生.pyc文件，只有在import相应的.py文件的时候，才会生成相应的.pyc文件

## 030、Python中pass、continue、break、return的区别

* pass： 不做任何事情，只是充当占位符的作用
* continue： 常用作循环中，表示跳出当前循环，执行下一次循环
* break： 常用作循环中，表示终止此次循环
* return：结束函数的运行，并返回结果

## 031、python中的异常处理模块

当程序出现错误，会终止运行。为了能够让程序依旧可以正常执行，我们引入了异常处理模块， 即：try...except... 其他的还有try..except..else 或try..except..finally

* try...except.. 捕获异常，使程序不会被终止

try:  
 num = 10 / 0  
except Exception as e:  
 print('分母不能为0')  
 # raise e # 可以选择将异常抛出

* try...except...else 如果没有发生异常，则执行else中的内容
* try:  
  num = 0 / 10  
  except Exception as e:  
  print('分母不能为0')  
  # raise e # 可以选择将异常抛出  
  else:  
  print('没有发生异常..')
* try...except...finally 不管有没有发生异常，都会执行finally中的内容
* try:  
  num = 0 / 10  
  except Exception as e:  
  print('分母不能为0')  
  # raise e # 可以选择将异常抛出  
  finally:  
  print('无论如何都会执行我')

## 032、python中有哪些标准异常类

|  |  |
| --- | --- |
| 异常名称 | 描述 |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| SystemExit | Python 解释器请求退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

## 033、列出Python中的几个魔法方法，并简述用途

\_\_ init\_\_:对象初始化方法

\_\_ new\_\_:创建对象时候执行的方法，单列模式会用到

\_\_ str\_\_:当使用print输出对象的时候，只要自己定义了**str**(self)方法，那么就会打印从在这个方法中return的数据

\_\_ del\_\_:删除对象执行的方法

## 034、sort和sorted的区别

* sort:在原有列表的基础上进行排序，无返回值
* arr = [-2, 1, 0, 7, -7, 4]  
  arr.sort()  
  print(arr)

-sorted: 排序会生成一个新的列表

arr = [-2, 1, 0, 7, -7, 4]  
arr\_sort = sorted(arr)  
print(arr) # [-2, 1, 0, 7, -7, 4]  
print(arr\_sort) # [-7, -2, 0, 1, 4, 7]

他们都有一个参数reverse,默认False，代表升序排序sorted(arr, reverse=False)

## 035、s="info:xiaoZhang 33 shandong",用正则切分字符串输出['info', 'xiaoZhang', '33', 'shandong']

|表示或，根据冒号或者空格切分

import re  
s = 'info:xiaoZhang 33 shandong'  
result = re.split(r':| ', s)  
print(result) # ['info', 'xiaoZhang', '33', 'shandong']

## 036、怎么样在函数内修改全局变量

修改全局变量的值可以使用global关键字。 global语句的使用方法很简单，基本格式是：关键字global，后跟一个或多个变量名

x = 10  
  
def func():  
 global x  
 x = 5  
 print(x)  
  
func()

## 037、简单描述一下你常用的五个标准库

* os：提供了不少与操作系统相关联的函数
* sys: 通常用于命令行参数
* re: 正则匹配
* math: 数学运算
* datetime:处理日期时间

## 038、简述一下Python中的GIL

GIL 是python的全局解释器锁，同一进程中假如有多个线程运行，一个线程在运行python程序的时候会霸占python解释器（加了一把锁即GIL），使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。如果线程运行过程中遇到耗时操作，则解释器锁解开，使其他线程运行。所以在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。

多进程中因为每个进程都能被系统分配资源，相当于每个进程有了一个python解释器，所以多进程可以实现多个进程的同时运行，缺点是进程系统资源开销大