1.基础知识

- 1.深拷贝和浅拷贝的区别是什么
- 2.解释一下 *args和**kwargs?
- 3.python和其他语言的对比
- 4.简述解释型和编译型编程语言
- 5.python解释器种类以及特点
- 6.位和字节的关系
- 7.ascii unicode utf-8 gbk区别
- 8.python2和python3的区别
- 9.文件操作时，xreadlines和readlines的区别
- 10.字符串 列表 元祖 字典每个常用的五个方法
- 11.lambda表达式格式以及应用场景
- 12.pandas中 map applymap and apply的区别
- 13.pass的作用
- 14.is和==的区别
- 15.python垃圾回收机制
- 16.python的可变类型和不可变类型
- 17.列举常见的内置函数
- 18.filter map reduce的作用
- 19.如何在函数中设置一个全局变量
- 20.logging模块的作用，应用场景
- 21.常用字符串格式化哪几种
- 22.简述生成器 迭代器 可迭代对象 以及应用场景
- 23.os和sys模块的作用
- 24.如何使用python删除一个文件
- 25.对面向对象的理解
- 26.python面向对象中的继承有什么特点
- 27.面向对象中super的作用
- 28.是否使用过functools中的函数，作用是什么?
- 29.列举带双下划綫的特殊方法：如new init
- 30.静态方法和类方法区别
- 31.列举面向对象中的特殊成员以及应用场景

1.基础知识

1.深拷贝和浅拷贝的区别是什么

深拷贝是将对象本身复制给另一个对象，是值拷贝，如果对对象的副本进行更改时不会影响原对象(copy.deepcopy())

- 两个变量的内存地址不同
- 两个变量各有各自的值，且互不影响
- 对其任意一个变量的值得改变不会影响另一个

浅拷贝是将对象的引用复制给另一个对象，是对内存地址的拷贝，如果对副本进行更改，将会影响原对象

- 公用一个值
- 两个变量的内存地址一样
- 对其中一个变量的值改变，另外一个变量的值也会改变

Python赋值过程中不明确区分拷贝和引用，一般对静态变量的传递为拷贝，对动态变量的传递为引用。（注，对静态变量首次传递时也是引用，当需要修改静态变量时，因为静态变量不能改变，所以需要生成一个新的空间存储数据）。

- 字符串，数值，元组均为静态变量
- 列表，字典为动态变量。

2.解释一下 *args和**kwargs?

如果我们不知道将多少个参数传递给函数，比如当我们想传递一个列表或者元祖值时，可以使用*args,*args用来将参数打包成tuple给函数体调用

当我们不知道将会传入多少关键字参数时，使用**kwargs会收集关键字参数，**kwargs打包关键字参数成dict给函数体调用

arg和*kwarg作用？

万能参数，解决了函数参数不固定的问题

*arg会把位置参数转化为tuple(非键值对的参数组）

**kwarg:会把关键字参数转化为dict，（键值对参数组）

3.python和其他语言的对比

python代码，简洁明确优雅，简单易懂

开发效率高，可扩展性强

4.简述解释型和编译型编程语言

解释型：在执行程序时，计算机才一条一条的将代码解释成机器语言给计算机来 执行

编译型：把源程序的每一条语句都编译成机器语言，并保存成二进制文件，这样计算机运行该程序时可以直接以机器语言来运行此程序，运行速度很快

5.python解释器种类以及特点

Cpython Ipython Jpython pypy Ironpython

python是一门解释器语言，代码想运行必须通过解释器执行，python存在多种解释器，分别基于不同语言开发，每个解释器有不同的特点，但都能正常运行python代码，常用五种解释器：

- Cpython:当从python官网下载并安装好python之后，直接获得了一个官方版本解释器cpython，是用c语言开发的，是使用最广的解释器
- Ipython，基于cpython之上的一个交互式解释器，只是在交互方式上有所增强，执行python代码的功能和cpython是一样的
- pypy，pypy是另一个python解释器，他的目标是执行速度，采用JIT技术，对python进行动态编译，所以可以显著提高速度
- jpython,java平台的解释器

6.位和字节的关系

一个字节=8位

7.ascii unicode utf-8 gbk区别

- ascii: 使用一个字节编码，范围基本只有英文字母 数字和一些特殊符号，只有256个字符
- unicode: 能够表示全世界所有的字节
- gbk:只用来编码汉字
- utf-8:一种针对unicode的可变长度字符编码，又称万国码

8.python2和python3的区别

- 默认编码：2-ascii 3-utf-8
- print的区别：python2里print是一个语句，不论想输出什么，直接放到print关键字后面即可；python3里，print()是一个函数，需要将输出的东西作为参数传给他
- 字符串：python2中有两种字符串类型：unicode字符串和非unicode字符串，python3中只有unicode字符串
- xrange(): python2里，有两种方法获得一定范围内的数字，range()返回一个列表，xrange()返回一个迭代器 python3里，只有range()返回迭代器，xrange不存在
- input的区别：python2中有两个输入，一个是input()，等待用户输入表达式，然后返回结果；另一个是raw_input()，输入什么返回什么。python3中只有input替代了他们
- 整数除法的（/)的区别：python2中两个整数相除，得到的结果也会是一个整数，将会把小数去掉，如3/2得1；3中两整数相除，如果有小数的话将会默认返回一个浮点数，如3/2返回1.5
- int和long的区别：2中分为int整型和long长整型，3中只有一种整数类型int
- import导入模块的搜索路径不同：2中采用import是采用相对路径方式进行搜索模块的，3采用的是绝对路径的方式进行import的
- 对缩进要求不同：2没有严格的缩进要求，3使用更加严格的缩进

9.文件操作时，xreadlines和readlines的区别

readlines返回一个list，xreadlines方法返回一个生成器

10.字符串 列表 元祖 字典每个常用的五个方法

字符串：replace strip split upper lower join find index

replace(old,new,[,max])
把字符串中的str1替换成str2,如果max指定，替换不超过max次

strip([chars]) 去空格及特殊符号
s.strip().lstrip().rstrip(',)

split(str="",num=string.count(str)) 以str为分隔符截取字符串，如果num有指定值，则仅截取num+1个子字符串

join(seq)
以指定字符串作为分隔符，将seq中所有的元素（的字符串表示）合并为一个新的字符串

find(str,beg=0,end=len(string))
检测Str是否包含在字符串中，如果指定范围，则检查是否包含在指定范围内，如果包含返回开始的索引值，否则返回-1

index(str,beg=0,end=len(string))
跟find方法一样，只不过如果Str不在字符串中会报一个异常

```
s1='strchr'
s2='s'
pos=s1.index(s2)
print(pos)
```

列表：append ,pop, insert,remove,reverse,sort,count,index...
增：
list.append(obj) 在列表末尾添加新的对象
list.insert(index,obj) 将对象插入列表

删：
list.pop([index=-1])
移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
list.remove(obj)
移除列表中某个值的第一个匹配项

改：
list.reverse() 反向列表中元素

查：
list.sort()(cmp=None,key=None,reverse=False) 对原列表进行排序
list.count(obj) 统计某个元素在列表中出现的次数

元祖：index count len() dir()
元祖不可更改所以只能查
max(tuple) 返回元祖中元素最大值

字典：get,keys,values,pop,popitems,clear,update,items.....

查：

radiansdict.get(key,default=None)　返回指定键的值，如果值不再字典中返回default值

radiansdict.keys()　返回一个迭代器，可以使用list()来转换为列表

radiansdict.values()　返回一个迭代器，可以使用list()来转换为列表

radiansdict.items()　以列表返回可遍历的（键，值）元祖数组

删：

pop(key,[,defalut])　删除字典给定键key所对应的值，返回值为被删除的值。key值必须给出

popitem()　随机返回并删除字典中的一对键和值

radiansdict.clear()　删除字典内所有元素

改：

radiansdict.update(dict2)　把字典dict2的键值对更新到dict里

```
dict={'Name':'Runoob','Age':7,'Class':,'First'}
dict['Age']=8
dict['School']="菜鸟教程"
```

11.lambda表达式格式以及应用场景

lambda函数赋值给其他函数

lambda后面跟一个或多个参数，紧跟一个冒号，以后是一个表达式。冒号前是参数，冒号后是返回值　lambda x:2x

12.pandas中 map applymap and apply的区别

apply()是一种让函数作用于列或者行（一维向量）操作　重点：选取数据的某行或者某列

applymap()是一种让函数作用于DataFrame每一个元素的操作（选取的是所有数据即Dataframe)

map是一种让函数作用于series每一个元素的操作

13.pass的作用

空语句 do nothing

保证格式完整

保证语义完整

14.is和==的区别

is判断内存地址是否相等，==判断数值是否相等

15.python垃圾回收机制

采用计数机制为主，标记-清楚和分代收集（隔代回收、分代回收）两种机制为辅的策略

计数机制

Python的GC模块主要运用了引用计数来跟踪和回收垃圾。在引用计数的基础上，还可以通过“标记-清除”解决容器对象可能产生的循环引用的问题。通过分代回收以空间换取时间进一步提高垃圾回收的效率。

标记-清除：

标记-清除的出现打破了循环引用，也就是它只关注那些可能会产生循环引用的对象

缺点：该机制所带来的额外操作和需要回收的内存块成正比。

隔代回收

原理：将系统中的所有内存块根据其存活时间划分为不同的集合，每一个集合就成为一个“代”，垃圾收集的频率随着“代”的存活时间的增大而减小。也就是说，活得越长的对象，就越不可能是垃圾，就应该减少对它的垃圾收集频率。那么如何来衡量这个存活时间：通常是利用几次垃圾收集动作来衡量，如果一个对象经过的垃圾收集次数越多，可以得出：该对象存活时间就越长。

16.python的可变类型和不可变类型

不可变类型：数字 字符串 元祖 不可变集合

可变类型：列表 字典 可变集合

17.列举常见的内置函数

map,filter（当返回值为正数时才返回函数结果进入下个元素的计算），zip,len,bin,oct

map() 会根据提供的函数对指定序列做映射。

第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

map(function, iterable, ...)

参数

- function – 函数
- iterable – 一个或多个序列

返回值

Python 2.x 返回列表。

Python 3.x 返回迭代器。

迭代器是访问集合元素的一种方式。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。

```
>>>def square(x) : # 计算平方数
... return x ** 2 ...
>>> map(square, [1,2,3,4,5]) # 计算列表各个元素的平方
[1, 4, 9, 16, 25]
>>> map(lambda x: x ** 2, [1, 2, 3, 4, 5]) # 使用 lambda 匿名函数
[1, 4, 9, 16, 25]
# 提供了两个列表，对相同位置的列表数据进行相加
>>> map(lambda x, y: x + y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10])
[3, 7, 11, 15, 19]
```

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 的元素放到新列表中。

参数

- function – 判断函数。
- iterable – 可迭代对象。

返回值

返回列表。

```
def is_odd(n):
return n % 2 == 1
newlist = filter(is_odd, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(newlist)
输出结果 :
[1, 3, 5, 7, 9]
```

zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。
如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用 * 号操作符，可以将元组解压为列表。
zip 语法：
zip([iterable, ...])
参数说明：
• iterabl – 一个或多个迭代器;
返回值
返回元组列表。

```
>>>a = [1,2,3]
>>> b = [4,5,6]
>>> c = [4,5,6,7,8]
>>> zipped = zip(a,b) # 打包为元组的列表 [(1, 4), (2, 5), (3, 6)]
>>> zip(a,c) # 元素个数与最短的列表一致 [(1, 4), (2, 5), (3, 6)]
>>> zip(*zipped) # 与 zip 相反，*zipped 可理解为解压，返回二维矩阵式 [(1, 2, 3), (4, 5, 6)]
```

bin() 返回一个整数 int 或者长整数 long int 的二进制表示.
参数
x – int 或者 long int 数字
返回值
字符串。

```
>>>bin(10)
'0b1010'
>>> bin(20)
'0b10100'
```

oct() 函数将一个整数转换成8进制字符串。

```
>>>oct(10)
'012'
>>> oct(20)
'024'
>>> oct(15)
'017'
>>>
```

hex() 函数用于将10进制整数转换成16进制，以字符串形式表示。

```
>>>hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
>>> hex(1L)
'0x1L'
>>> hex(12)
'0xc'
>>> type(hex(12))
<class 'str'> # 字符串
```

hash() 用于获取取一个对象（字符串或者数值等）的哈希值。
hash 语法：
hash(object)
参数说明：
• object – 对象;
返回值
返回对象的哈希值。

enumerate(sequence, [start=0])

enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

参数

sequence – 一个序列、迭代器或其他支持迭代对象。

start – 下标起始位置。

返回值

返回 enumerate(枚举) 对象。

```
for 循环使用 enumerate
>>>seq = ['one', 'two', 'three']
>>> for i, element in enumerate(seq):
... print i, element
...
0 one
1 two
2 three
```

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

eval() 函数用来执行一个字符串表达式，并返回表达式的值。

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
>>> eval( '3 * x' )
21
>>> eval('pow(2,2)')
4
```

str.format()

格式化字符串的函数 str.format()，它增强了字符串格式化的功能。

基本语法是通过 {} 和 : 来代替以前的 % 。

format 函数可以接受不限个参数，位置可以不按顺序。

```
实例
>>>"{} {}".format("hello", "world") # 不设置指定位置，按默认顺序 'hello world' >>> "{0} {1}".form
>>> "{1} {0} {1}".format("hello", "world") # 设置指定位置 'world hello world'
```

reduce(function, iterable[, initializer])

reduce() 函数会对参数序列中元素进行累积。

函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

参数

- function – 函数，有两个参数
- iterable – 可迭代对象
- initializer – 可选，初始参数

返回值

返回函数计算结果。

```
实例
以下实例展示了 reduce() 的使用方法：
>>>def add(x, y) : # 两数相加
... return x + y ...
>>> reduce(add, [1,2,3,4,5]) # 计算列表和：1+2+3+4+5
15
>>> reduce(lambda x, y: x+y, [1,2,3,4,5]) # 使用 lambda 匿名函数
15
```

18.filter map reduce的作用

filter(function,iterable) 过滤函数

map(function,iterable) 循环函数

reduce(function,iterable)累积函数

19.如何在函数中设置一个全局变量

内置语法 globals 全局变量

20.logging模块的作用，应用场景

程序调试

了解软件程序运行情况，是否正常

软件程序运行故障分析与问题定位

21.常用字符串格式化哪几种

1. %s %

```
print("这位同学叫%s,年龄是%d岁" %('小明',10))
```

2.format格式化输出

```
print("这位同学叫{name},年龄是{age}岁".format(name="小明",age=18))
```

3.print(f'内容{变量名}')

22.简述生成器 迭代器 可迭代对象 以及应用场景

生成器：在python中，一边循环一边计算的机制，称为生成(generator)，通过next()取值，两种表现形式

1.将列表生成式的[]改为()

2.含有yield关键字的函数

应用场景：优化代码，节省内存。比如使用python读取一个10g的文件，如果一次性将10g的文件加载到内存处理的话（read方法），内存肯定会溢出；这里如果可以使用生成器把读写交叉处理进行，比如使用(readline和readlines)就可以在循环读取的同时不断处理，可以节省大量的内存空间。

带有yield的函数在python中被称为生成器

迭代器：是访问集合元素的一种方式。迭代器同时实现了_iter_和_next_方法

可迭代对象：只要实现了_iter_方法的对象就是可迭代对象，可以用for循环语句遍历的对象就是可迭代对象

23.os和sys模块的作用

os模块负责程序与操作系统的交互，提供了访问操作系统底层的接口

sys模块负责程序与python解释器的交互，提供了一系列的函数和变量，用于操控python的运行时环境

24.如何使用python删除一个文件

```
import os
os.remove(r'path'))
```

25.对面向对象的理解

面向对象的程序设计的核心是对象，对象是特征和技能的结合，其中特征和技能分别对应对象的数据属性和方法属性。

面向对象编程可以将数据与函数绑定到一起，进行封装，这样能快速的开发程序，减少了重复代码的重写过程。

优点是：解决了程序的扩展性。对某一个对象单独修改，会立刻反映到整个体系中，如对游戏中一个任务参数的特征和技能修改都很容易

缺点：可控性差，无法像面向过程的程序设计流水线式的可以很精准的预测问题的处理流程与结果

应用场景：需求经常变化的软件，一般需求的变化都集中在用户层，互联网应用，企业内部软件，游戏等都是面向对象的程序设计大显身手的好地方

26.python面向对象中的继承有什么特点

- 在继承中基类的构造（init()方法）不会被自动调用，它需要在其派生类的构造中亲自专门调用
- python同时支持单继承与多继承，当只有一个父类时为单继承，存在多个父类时为多继承。并且子类会继承父类的所有的属性和方法，子类也可以覆盖父类同名的变量和方法。
- 在调用基类的方法时，需要加上基类的类名前缀，且需要带上self参数变量。区别于在类中调用普通函数时并不需要带上self参数
- python总是首先查找对应类型的方法，如果它不能再派生类中找到对应的方法，它才开始到基类中逐个查找（先在本类中查找调用的方法，找不到才去基类中找）

27.面向对象中super的作用

- super在面向对象继承类中代指父类，可以快速调用父类，书写方法super(子类名，self).属性或者方法或super().属性或者方法
- super方法可以增加类之间调用的灵活性，当父类名发生变化时不必修改
- super方法在类的多继承时可以简化代码，避免代码冗余
- super机制里可以保证公共父类仅被执行一次，执行的顺序遵循MRO，广度优先查询方法

28.是否使用过functools中的函数，作用是什么?

functools用于高阶函数，指那些作用于函数或者返回其他函数的函数。通常情况下，只要是可以被当做函数调用的对象就是这个模块的目标

29.列举带双下划綫的特殊方法：如new init

new:构造方法，创建一个对象，实例化时第一个被执行，返回一个创建好的对象及_init_(self)的self，只有继承了object的类才会有这个方法

init:初始化方法，_init_在_new_的基础上完成一些其他初始化的动作，_init_没有返回值

30.静态方法和类方法区别

静态方法：既不使用类中的属性又不使用对象中的属性，由类或者对象调用的方法，依赖pythonzhuangshiqi@staticmethod来实现

类方法：只使用类中的静态变量，一般都是由类调用，依赖python装饰器@classmethod来实现

31.列举面向对象中的特殊成员以及应用场景

- call:对象的构造方法，对象加上()，可以触发这个类的_call_方法
- len:内置函数的len函数是依赖类中的_len_方法
- eq:判断值是否相等的时候依赖_eq_方法
- hash:判断hash值是否相等的时候依赖_hash_方法（拓展：set的去重机制其实就是根据_hash_和_eq_方法实现的）
- str:和str() print() %s都是息息相关的，返回值一定是字符串类型
- repr:和repr() %r都是息息相关的，在没有_str_方法时，repr_可以完全取代_str
- del析构方法，对应着一个对象的删除之前执行的内容

32.ininstance作用以及应用场景

判断一个对象是否是一个已知的类型

33.异常处理写法以及如何主动抛出异常（应用场景）

异常处理的常规写法：

```
try:
    #执行的主题函数
except Exception as e:
    print(str(e))
```

主动抛出异常：

```
raise TypeError('出现了不可思议的异常') #TypeError可以是任意的错误类型
```

34.简述yield 和 yield from关键字

yield是一个类似return的关键字，只是这个函数返回的是个生成器，当你调用这个函数的时候，函数内部的代码并不立马执行，这个函数只是返回一个生成器对象，当你使用for进行迭代的时候，函数中的代码才会执行。

yield关键字的工作机制：如果函数里面有yield关键字，这个函数的返回值是生成器。如果遇到yield，函数停止执行，当再次调用next方法时，从停止的地方继续执行。默认next方法会把yield后面的值返回回来。

yield from的主要功能是打开双向通道，把最外层的调用方与最内层的子生成器连接起来，这样二者可以直接发送和产出值，还可以直接传入异常，而不用在位于中间的协程中添加大量异常处理的样板代码。

有了这个结构，协程可以通过以前不可能的方式委托职责。

35.解释python中的三元表达式

与c++不同，在python中我们不需要使用?符号，而是使用如下语法：

[on true] if [expression] else [on false]

如果[expression]为真，则[on true]部分被执行。如果表示为假则[on false]部分被执行

36.解释继承

一个类继承自另一个类，也可以说是一个孩子类/派生类/子类，继承自父类/基类/超类，同时获取所有的类成员（属性和方法）

继承可以使我们重用代码，还可以更方便地创建和维护代码。支持以下类型的继承：

- 单继承-一个子类类继承自单个基类
- 多重继承-一个子类继承自多个基类
- 多级继承-一个子类继承自一个基类，而基类继承自另一个基类
- 分层继承-多个子类继承自同一个基类
- 混合继承-两种或两种以上类型继承的组合

37.简单谈谈装饰器的作用和功能

装饰器本质上是一个函数，该函数用来处理其他函数，它可以让其他函数在不需要修改代码的前提下增加额外的功能，装饰器的返回值也是一个函数对象。写代码要遵循 开放封闭 原则，简单来说，它规定已经实现的功能代码不允许被修改，但可以被扩展。

装饰器具体可以实现以下功能：

1. 引入日志
2. 函数执行时间统计
3. 执行函数前预备处理
4. 执行函数后清理功能
5. 权限校验等场景
6. 缓存

38.python中的模块和包是什么

模块就是工具包，要想使用这个工具包中的工具，就需要导入这个模块。模块是非常简单的python文件，单个python文件就是一个模块。

包就是将有联系的模块组织在一起的一个集合，有效避免名称冲突问题，让应用组织结构更加清晰。

39.python是如何进行内存管理的 （如何实现垃圾回收机制的）

垃圾回收机制采用的是引用计数机制为主，标记-清楚和分代收集两种机制为辅的策略。

40.面向对象深度优先和广度优先是什么，说明应用场景

1. 深度优先：经典类多继承搜索的顺序，先深入继承树左侧查找，然后再返回，开始查找右侧
2. 广度优先：新式类多继承搜索的顺序，先在水平方向查找，然后再向上查找

41.简述_init_和_len_这两个魔术方法的作用

构造方法_init_：当一个对象被创建后，会立即调用该构造方法，自动执行构造方法里面的内容

构造方法_len_:该构造方法会返回元素的数量

42.set和list的相互转换

set转成list方法如下：

```
s = set('12342212')
print s    # set(['1', '3', '2', '4'])
l = list(s)
l.sort()    # 排序
print l    # ['1', '2', '3', '4']
```

list转成set方法如下：

```
l = ['12342212']
s = set(l[0])
  print s    # set(['1', '3', '2', '4'])
m = ['11','22','33','44','11','22']
 print set(m)    # set(['11', '33', '44', '22'])
```