5/25/2023

执行下列选项代码，输出[1, {'age': 10}]的是（）D

A

a = [1,{'age':10}]

b = a

a[1]['age'] = 12

print(b)

B

a = [1,{'age':10}]

b = a[:]

a[1]['age'] = 12

print(b)

C

a = [1,{'age':10}]

b = a.copy()

a[1]['age'] = 12

print(b)

D

import copy

a = [1,{'age':10}]

b = copy.deepcopy(a)

a[1]['age'] = 12

print(b)

A选项是值复制，修改变量a的可变对象和不可变对象都会对变量b产生影响，BC选项是浅复制，修改变量a的可变对象会对b产生影响，D选项是深复制，修改可变和不可变对象对b都无影响

执行以下程序，输出结果为（）True

a = 0 or 1 and True

print(a)

0的布尔值为False，所以0 or 1的结果为1,1的布尔值为True，所以1 and True的结果为True

dicts = {'a': 1, 'b': 2, 'c': 3}

print(dicts.pop())

报错

python3中，关于字典的pop方法需要指定删除的key

def bar(multiple):

def foo(n):

return multiple \*\* n

return foo

\*\*是取指数

bar()()

一般而言，调用一个函数是加一个括号。如果看见括号后还有一个括号，说明第一个bar函数返回了一个函数，如果后面还有括号，说明前面那个也返回了一个函数。以此类推

strs = 'abbacabb'

print(strs.strip('ab'))

python3中，strip方法匹配两侧所有的不符合条件的字符（括号内指定字符串中的每个字符）

strs.strip('ab')

'ab'表示的是一种集合，这里是指：[ab,ba,aa,bb,aaa,bbb,abb,baa]等;

strs两端，只要是包含了上述集合中的任何一个，都删除

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

print list[10:]

输出[]

如果是索引访问，会导致IndexError，但本体考的是切片，切片操作时如果索引越界不会导致IndexError，只是返回一个空序列，这里返回空列表 []

执行以下程序，下列选项中，说法正确的是（）

tup = (1,2,[3,4]) ①

tup[2]+=[5,6] ②

执行代码②时会抛出异常，最终tup的值为(1,2,[3,4,5,6])

tup[2]是列表，列表是可变对象，对于可变对象，执行+=后并不会改变其id值，A错误；执行②时会抛出异常，“+=”的执行顺序是先对[3,4]执行+操作，其结果为[3,4,5,6]，然后再执行”=“，此时会抛出异常，这是因为元组不允许元素的引用被重新赋值，尽管赋值后列表的id是不变的，若改成tup[2].extend([5,6])就不会抛出异常

函数可以赋值给一个变量

B

函数可以作为元素添加到集合对象中

C

函数可以作为参数值传递给其它函数

D

函数可以当做函数的返回值

print type(1/2) <type 'int'>

列表lis=[1,2,3,4,5,6]，其切片lis[-1:1:-1]结果为[6,5,4,3]

切片[start:stop:step]，当step为负值时，表示从右往左，索引start=-1的元素是6，索引stop=1的元素是2，但是不会取到该索引，因此从右往左，间隔为1取到的结果就是D选项

自己运行自己的程序，\_\_name\_\_ 即为\_\_main\_\_

当程序当做脚本被别人调用运行时，其\_\_name\_\_ 为自己的模块名称，这里的print\_func即被导入到print\_module中当做脚本被运行时，则print\_func的\_\_name\_\_即变为其自身的模块名。

zip函数：

>>>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)]

fromkeys() 方法 ：

fromkeys() 函数用于创建一个新字典，以序列 seq 中元素做字典的键，value 为字典所有键对应的初始值

\*\*方法只用来创建新字典，不负责保存。当通过一个字典来调用 fromkeys 方法时，如果需要后续使用一定记得给他复制给其他的变量

eg：

>>> dict1={}

>>> dict1.fromkeys((1,2,3),'number')

{1: 'number', 2: 'number', 3: 'number'}

>>> print(dict1)

{}

>>> dict2=dict1.fromkeys((1,2,3),'number')

>>> dict2

{1: 'number', 2: 'number', 3: 'number'}

为输出一个字典dic = {‘a’:1,'b':2}

lis1 = ['a','b']

lis2 = [1,2]

dic = dict(zip(lis1,lis2))

print(dic)

dic = dict(a=1,b=2)

print(dic)

lis = ['a','b']

dic = dict.fromkeys(lis)

dic['a'] = 1

dic['b'] = 2

print(dic)

在Python3中，字符串的strip()函数为删除字符串的前后空格，rstrip() 为删除字符串末尾的空格

strs = ' I like python '

one = strs.strip()

print(one)

two = strs.rstrip()

print(two)

'I like python', ' I like python'

A

b = 1

def fn():

nonlocal b

b = b + 1

print(b)

fn()

B

tup = (('onion','apple'),('tomato','pear'))

for \_,fruit in tup:

print(fruit)

C

a = [b for b in range(10) if b % 2 == 0]

print(b)

D

lis = [1,2,'a',[1,2]]

set(lis)

A选项，变量b属于全局变量，所以应该使用global声明而不是nonlocal；B选项，可以使用\_作为占位符，所以B选项不会报错；C选项，python3.x中，使用列表生成式，中间变量b在生成列表后被销毁，因此再次使用变量b会报错；D选项，集合元素不能是列表这类不可散列对象，因此会报错。

kvps = { '1' : 1, '2' : 2 }

theCopy = kvps.copy()

kvps['1'] = 5

sum = kvps['1'] + theCopy['1']

print sum

6

这里的 kvps.copy()指的是浅拷贝，kvps['1'] 和theCopy['1']原本指向的都是1，但是因为int是不可变的的，所以kvps['1']=5，使得kvps['1']更改指向了，但是theCopy指向的1是int，本身没有改变也不能改变，所以还是1

strs = 'abcd'

print(strs.center(10, '\*'))

'\*\*\*abcd\*\*\*'

在Python3中，center() 返回一个原字符串居中,并使用空格填充至长度 width 的新字符串。默认填充字符为空格。题目中填充长度 width=10, 填充字符为 ‘\*’，最终的结果为 '\*\*\*abcd\*\*\*'

执行以下程序，输出结果为（）

def fun(a=(),b=[]):

a += (1,)

b.append(1)

return a,b

fun()

print(fun())

((1,), [1, 1])

第一个默认参数是不可变对象，第二个参数是可变对象

当执行第一个fun()时，先实例化了 a对象和b对象然后 a=(1,0) b=[1]

当执行第二个fun()时，a是元组是不可变对象所以还是a=(1) 但是b因为是可变对象所以会在原来的内存空间变为[1,1]

5/26/2023

dicts = {}

dicts[([1, 2])] = 'abc'

print(dicts)

在Python3中，只有当元组内的所有元素都为不可变类型的时候，才能成为字典的key，因此程序运行过程中会报错：TypeError: unhashable type: 'list'

对列表a = [1,2,3,1,2,4,6]进行去重后，得到列表b，在不考虑列表元素的排列顺序的前提下，下列方法错误的是

a.sort()

for i in range(len(a)-1):

if a[i] == a[i+1]:

a.remove(a[i])

else:

continue

b = a

错误原因在于for循环的计数次数是不变的，但是随着a重复元素不断的移除，会导致列表出现IndexError

def dec(f):

n = 3

def wrapper(\*args,\*\*kw):

return f(\*args,\*\*kw) \* n

return wrapper

@dec

def foo(n):

return n \* 2

foo(2) == 12

A screenshot of a computer program

Description automatically generated with medium confidence

strs = ' I like python '

one = strs.split(' ')

two = strs.strip()

print(one)

print(two)

['', 'I', 'like', 'python', '']，'I like python'

有一段python的编码程序如下,请问经过该编码的字符串的解码顺序是（ ）

1

urllib.quote(line.decode("gbk").encode("utf-16"))

题目中的代码是一个编码过程：

编码:decode()

解码:encode()

url编码:urllib.quote()

line.decode("gbk") 可知 line 是 gbk 编码的

编码过程：

line -> 解码 gbk -> 编码 utf-16 -> 编码 url

解码过程（与编码过程相反）：

解码 url -> utf-16 -> gbk

lists = [1, 2, 3, 4]

tmp = 0

for i,j in enumerate(lists):

tmp += i \* j

print(tmp)

i j

0 1

1 2

2 3

3 4

其中i为索引，j为数据对象list中元素，所以tmp = 0x1+1x2+2x3+3x4 = 20

names = ["Andrea", "Aaslay", "Steven", "Joa"]

lists = []

for name in names:

if name.count('a') >= 2:

lists.append(name)

print(lists)

['Aaslay']

def chanageList(nums):

nums.append('c')

print("nums", nums)

str1 = ['a', 'b']

# 调用函数

chanageList(str1)

print("str1", str1)

nums ['a', 'b', 'c']，str1 ['a', 'b', 'c']

在调用chanageList()函数时，str1 作为参数传入该函数后执行 append()函数添加字符 ‘c’，虽然传递给变量nums但其表示的就是str1，因此A正确

python变量的查找顺序为（）

局部作用域>外部嵌套作用域>全局作用域>内置模块作用域

info = {'name':'班长', 'id':100, 'sex':'f', 'address':'北京'}

age = info.get('age')

print(age)

age=info.get('age',18)

print(age)

None 18

获取字典dict中key的值：dict.get(key,default=None），如果key在dict中则返回对应的值，否则返回default的值，其中default的值可自己指定

import math

def sieve(size):

sieve= [True] \* size

sieve[0] = False

sieve[1] = False

for i in range(2, int(math.sqrt(size)) + 1):

k= i \* 2

while k < size:

sieve[k] = False

k += i

return sum(1 for x in sieve if x)

print(sieve(10000000000))

455052511

求100亿以内的质数个数

A

\_\_new\_\_是一个静态方法，而\_\_init\_\_是一个实例方法

B

\_\_new\_\_方法会返回一个创建的实例，而\_\_init\_\_什么都不返回

C

只有在\_\_new\_\_返回一个cls的实例时，后面的\_\_init\_\_才能被调用

D

当创建一个新实例时调用\_\_new\_\_，初始化一个实例时用\_\_init\_\_

根据官方文档：

\_\_init\_\_是当实例对象创建完成后被调用的，然后设置对象属性的一些初始值。

\_\_new\_\_是在实例创建之前被调用的，因为它的任务就是创建实例然后返回该实例，是个静态方法。

即，\_\_new\_\_在\_\_init\_\_之前被调用，\_\_new\_\_的返回值（实例）将传递给\_\_init\_\_方法的第一个参数，然后\_\_init\_\_给这个实例设置一些参数

a = '123'

b = '123'

a,b为字符串不可变类型，所以指向相同地址，所以 a is b

is指地址相同

==内容相同

===内容和格式相同

a+b=‘123123’

错选a==123，字符和int不相同

dicts = {}

dicts[(1, 2)] = ({3, (4, 5)})

print(dicts)

第一个是 字典的格式就是 dict[key] = value ， 由题目可知 key 是：(1,2) value是：({3, (4, 5)}）

关于value的值 （{3, (4, 5)}） ， 因为它不是元组，所以把括号去掉：{3, (4, 5)}， 它是一个set，set 内部会自动排序， 元组在set里面会往前放，所以变成：{(4, 5), 3}。

所以最后的结果 key：value = （1，2）：{（4，5），3}

num = 1

def fn():

num += 1

return lambda:print(num)

x = fn()

x()

函数内部修改同名全局变量需要使用global关键字声明，即global num num+=1，否则报UnboundLocalError

python搜索模块的顺序为:内建模块>当前路径，即执行Python脚本文件所在的路径>环境变量中的PYTHONPATH>python安装路径

5/27/2023

在python中，使用open方法打开文件,语法如下：

open(文件名，访问模式)

如果以二进制格式打开一个文件用于追加，则访问模式为ab

"r","w","a","rb","wb","ab"：指定对文件打开方式即文件内容操作方式，即只读，可写，追加，二进制读，二进制写，二进制追加

1

2

strs = 'abcd12efg'

print(strs.upper().title())

'Abcd12Efg'

在Python3中，upper() 表示将字符小写转换为大写，title方法将字符串内每个连续字母的首字母大写，其余小写，数字后面的字母被title当成首字母大写了

1、\*args和\*\*kwargs 这两个是python中方法的可变参数。

2、\*args表示任何多个无名参数，它是一个tuple；

3、\*\*kwargs表示关键字参数，它是一个dict。并且同时使用\*args和\*\*kwargs时，必须\*args参数列要在\*\*kwargs前，像foo(a=1, b='2', c=3, a', 1, None, )这样调用的话，会提示语法错误“SyntaxError: non-keyword arg after keyword arg”。

4、当方法的参数不确定时，可以使用\*args 和\*\*kwargs

def fun(a,\*,b):

print(b)

fun(1,2,3,4)

报错

如果你希望强迫调用者使用某些参数，且必须以关键字参数的形式传参，那么你只需要在所需位置的前一个位置放置一个\*。 def f1(a, \*, b, c): return a + b + c 对于上面这个函数而言，调用时参数a可以任意值, 但b,c参数一定要以关键字参数的形式传参，如f1(1, b=4, c=5),否则将会报错。

truple = (1, 2, 3)

print(truple\*2)

(1, 2, 3, 1, 2, 3)

在python3中，元组的 “\*” 运算也表示元组复制组合，最后会生成一个新的元组：(1, 2, 3, 1, 2, 3)

print r"\nwoow"

the text like exactly like this: \nwoow

Python 中字符串的前导 r 代表原始字符串标识符，该字符串中的特殊符号不会被转义，适用于正则表达式中繁杂的特殊符号表示。

最典型的例子，如要输出字符串 \n，由于反斜杠的转义，因此一般的输出语句为：

print "\\n"

这里的 \\ 将被转义为 \ 。而采用原始字符串输出时，则不会对字符串进行转义：

print r"\n"

因此本题答案为 C，输出 \nwoow 。注意前导标识符 r 不会被输出，只起标记作用

print('\n'.join(['a', 'b', 'c']))

在Python3中，join会把换行符当成普通连接字符进行连接,通过换行连接，因此

print('\n'.join(['a', 'b', 'c']))

a

b

c

集合不能索引取值

Python3 字典 fromkeys(seq, value) 函数用于创建一个新字典，以序列 seq 中元素做字典的键，value 为字典所有键对应的初始值。该方法返回一个新字典

tmp = dict.fromkeys(['a', 'b'], 4)

print(tmp)

{'a': 4, 'b': 4}

count()函数没有匹配到对象返回0

index()函数没有匹配到对象报错value Error

find()函数没有匹配到对象返回-1

in 没有匹配到对象返回false

def test(a, b, \*args):

print(a)

print(b)

print(args)

test(11, 22, 33, 44, 55, 66, 77, 88, 99)

11 22 (33, 44, 55, 66, 77, 88, 99)

def test(a,b,\*args)

a,b位置参数；\*args为不定长位置参数，传入的参数被放入一个元祖里。

所以，a,b分别被赋值11、22，而其他的数被存到一个元组里

已知a = [1, 2, 3]和b = [1, 2, 4]，那么id(a[1])==id(b[1])的执行结果 （）True

python中对于小整数对象有一个小整数对象池，范围在[-5,257）之间。对于在这个范围内的征数，不会新建对象，直接从小整数对象池中取就行

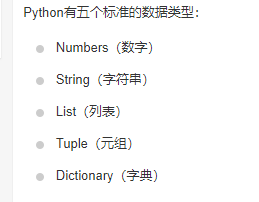
lists = [1, 1, 2, 3, 4, 5, 6]

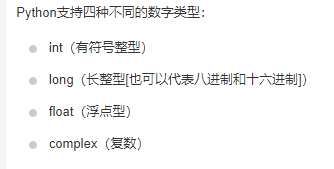
lists.remove(1)

lists.extend([7,8,9])

print(lists)

在Python3中list.remove(obj)移除列表中某个值的第一个匹配项， list.extend(seq)在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表），所以 lists.remove(1) 执行之后 lists = [1,2,3,4,5,6]，再执行 lists.extend([7,8,9]) 后lists = [1,2,3,4,5,6,7,8,9]





import random

def foo(n):

random.seed()

c1 = 0

c2 = 0

for i in range(n):

x = random.random()

y = random.random()

r1 = x \* x + y \* y

r2 = (1 - x) \* (1 - x) + (1 - y) \* (1 - y)

if r1 <= 1 and r2 <= 1:

c1 += 1

else:

c2 += 1

return c1 / c2

A picture containing text, handwriting, ink, paper

Description automatically generated

1.表示复数的语法是real + image j

2.实部和虚部都是浮点数

3.虚部的后缀可以是 “j” 或者 “J”

4.复数的 conjugate 方法可以返回该复数的共轭复数

\_foo 不能直接用于’from module import \*’

B

\_\_foo解析器用\_classname\_\_foo来代替这个名字，以区别和其他类相同的命名

C

\_\_foo\_\_代表python里特殊方法专用的标识

strs = 'I like python'

one = strs.find('a')

print(one)

two = strs.index('a')

print(two)

-1， 报错

在Python3中，string.find(str, beg=0, end=len(string))，检测 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，如果存在返回第一次出现位置的索引值，否则返回-1，

string.index(str, beg=0, end=len(string))，跟find()方法一样，只不过如果str不在 string中会报一个异常。

执行下列选项的程序，输出结果为True的是

lis = [1,3,2]

a = id(lis)

lis += [4,5]

b = id(lis)

print(a==b)

Python 中的 tuple 结构为 “不可变序列”，用小括号表示。为了区别数学中表示优先级的小括号，当 tuple 中只含一个元素时，需要在元素后加上逗号

one = (1, 2, 3)

two = ('a', 'b')

print(one+two)

在python3中，元组的“+”运算表示连接，因此程序结果会生成一个新的元组：(1, 2, 3, 'a', 'b')

执行以下程序，当用户输入0时，输出结果为（）

dividend = 1

divide = int(input())

try:

result = dividend / divide

print(1,end=" ")

except ZeroDivisionError:

print(2,end=" ")

except Exception:

print(3,end=" ")

else:

print(4)

2

在try...except...else结构中，当执行try程序块的语句时，若出现异常的语句，则不会继续执行try还未执行的代码，而是直接跳到except程序块，由于0不能作为分母，其抛出的异常对象属于异常类ZeroDivisionError，结果输出2，当异常被处理完时，会直接跳出except程序块，当try程序块没有出现异常时，不会执行except而执行else语句，出现异常时则执行except而不执行else，所以最终输出结果是2

在任何计算机上，Python3代码中的float类型都没有办法直接表示[0,1]区间内的所有实数。