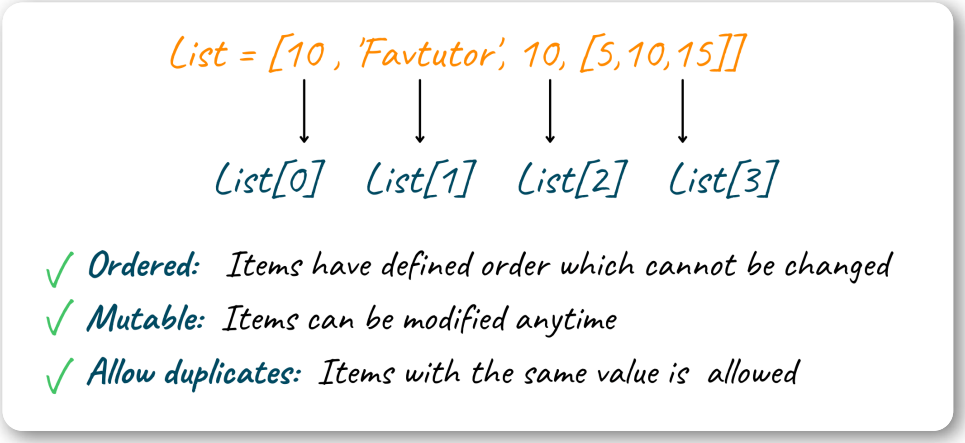
Python列表



列表用于存储任意数目、任意类型的数据集合。在Python中，用方括号（[]）来表示列表，并用逗号来分隔其中的元素。

> list=[元素1,元素2,...]

列表的创建

```
aList=[2,3,1,1,2,3,'a string']
aList

aList=list((2,3,1,1,2,3,'a string'))
aList

list=list((2,3,1,1,2,3,'a string')) #需要注意变量名字
list

aList=list(range(3,-10,-1))
aList
```

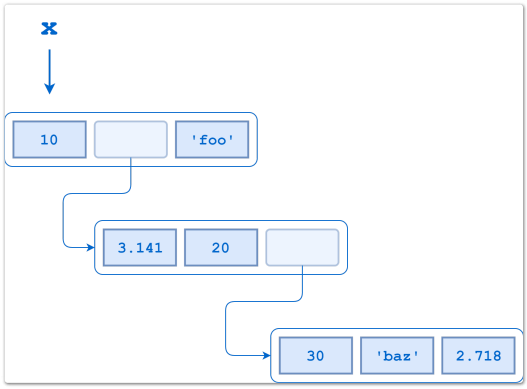
列表生成式

```
aList=[x for x in 'abcdefg']
aList

aList=[]
for x in 'abcdefg':
    aList.append(x)
aList

aList=[x*2 for x in range(1,5)]
aList

aList=[x*2 for x in range(1,20) if x%5==0 ]
aList
```

实时效果反馈

1. 哪一个选型可以从嵌套列表中将'baz'的'z'返回:

```
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```



A x[1][2]

B x[1][2][1]

C x[1][2][1][2]

D 以上都不是

2. 运行下面代码的结果是：

```
a = ['foo', 'bar', 'baz', 'qux', 'quux',
'corge']
a[-6]='python'
print(a[-6])
```

A IndexError: list index out of range

B corge

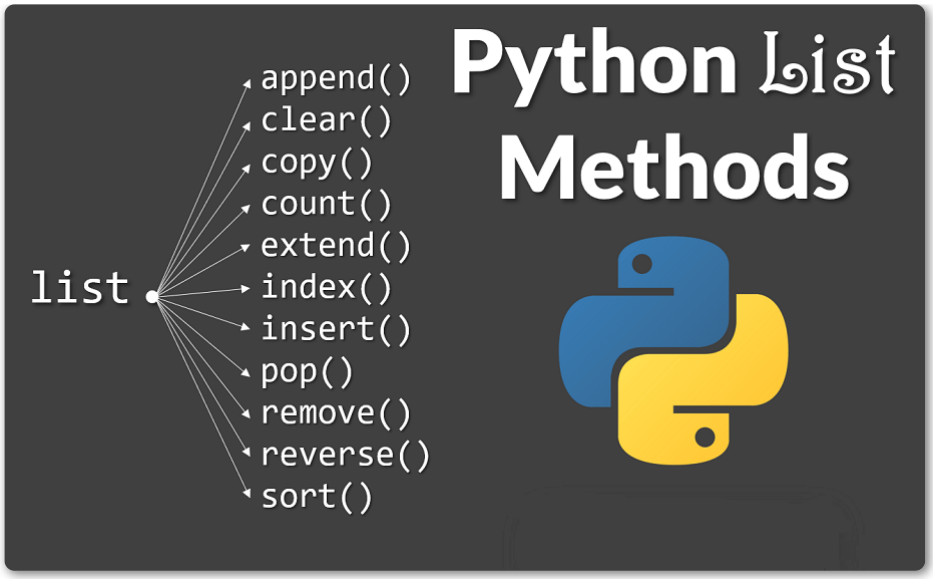
C foo

D python

答案

1=>C　　2=>D

Python列表基本操作



在列表的末尾追加一个新对象，使用append()方法。

```
#列表中添加元素
a=[10,20,30]
a.append(40)
print('增加元素后的列表：',a)
```

+运算符操作，并不是真正的尾部添加元素，而是创建新的列表对象；将原列表的元素和新列表的元素依次复制到新的列表对象中。这样，会涉及大量的复制操作，对于操作大量元素不建议使用。

```
a=[10,20,30]
b=[40,50]
a+b
```

extend()方法，将目标列表的所有元素添加到本列表的尾部，属于原地操作，不创建新的列表对象。

```
a=[10,20,30]
b=[40,50]
a.extend(b)
print(a)
```

使用insert()方法可以将指定的元素插入到列表对象的任意制定位置。这样会让插入位置后面所有的元素进行移动，会影响处理速度。涉及大量元素时，尽量避免使用。类似发生这种移动的函数还有：remove()、pop()、del()，它们在删除非尾部元素时也会发生操作位置后面元素的移动。

```
#使用insert函数插入元素
a=[10,20,30]
a.insert(2,100)          #在列表a的索引2处插入元素100
print(a)
```

index()可以获取指定元素首次出现的索引位置。语法是：index(value,[start,[end]])。其中，start和end指定了搜索的范围。

```
a = [10,20,30,40,50,20,30,20,30]
print(a.index(20))       #从列表中搜索第一个20
print(a.index(20,3))     #从索引位置3开始往后搜索的第一个20
```

del删除列表指定位置的元素。语法格式：del 元素。

```
a = [10,20,30,40,50,20,30,20,30]
a[1] = 5000
print(a)
del a[1]
print(a)
del a
print(a)
```

> pop()删除并返回指定位置元素，如果未指定位置则默认操作列表最后一个元素。

```
a=[1,2,3,4,5,6]
b=a.pop()          #没有指定位置，默认的是最后一个元素
print(b)
print(a)
c=a.pop(2)         #删除下标为2的元素
print(c)
print(a)
```

> remove删除首次出现的指定元素，若不存在该元素抛出异常。

```
a = [10,20,30,40,50,20,30,20,30]
a.remove(20)
print(a)
a.remove(100)                  #没有100这个元素，会抛出异常
```

列表乘以一个数字n会生成新的列表，在新的列表中原来的列表将会被重复n次。

```
a = ['sxt',100]
b = a*3
print(a)
print(b)
```

len、max和min这三个函数用于返回列表中元素的数量、列表中最大值、列表中最小值。使用max和min函数要注意一点，就是列表中的每个元素值必须是可比较的。否则会抛出异常。例如，如果列表中同事包含整数和字符串类型的元素，那么使用max、min函数就会抛出异常。

```
a=[10,20,30,40,50,60]
print(len(a))        #运行结果是6
print(max(a))        #运行结果是60
print(min(a))        #运行结果是10
b=['a',30,'b',40]
print(max(b))         #字符串和数字不能比较，将抛出异常
```

列表count()方法的使用

```
#count()统计元素出现的次数
a=[10,20,40,30,10,20,50,10]
print('元素10出现的次数：',a.count(10))
print('元素20出现的次数：',a.count(20))
print('元素30出现的次数：',a.count(30))
```

列表reverse()方法的使用

```
#reverse()用于将列表中的元素反向存放
a=[1,2,3,4,5,6,7]
a.reverse()
print(a)
```

列表sort ()、sorted方法的使用

```
#sort()用于对列表进行排序，调用该方法会改变原来的列表
a=[11,20,13,34,5,36,17]
a.sort()
print(a)
print('正序：',a)
a.sort(reverse=True)
print('逆序：',a)

#sorted用于对列表进行排序，生成新列表，不改变原来的列表
print('-'*5,'sorted排序','-'*5)
a=[11,20,13,34,5,36,17]
b=sorted(a)
```

```
13 print('a列表：',a)                    #原来列表不会被修改
14 print('正序b列表：',b)
```

实时效果反馈

1. 下列哪一种方式不能将列表扩充为['a','b','c','d','e']:

```
1 a = ['a','b','c']
```

A a.extend(['d', 'e'])

B a += ['d', 'e']

C a[len(a):] = ['d', 'e']

D a.append(['d', 'e'])

2. 下列哪一种方式不能将列表中的元素3删除：

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

A del a[2]

B a.remove(3)

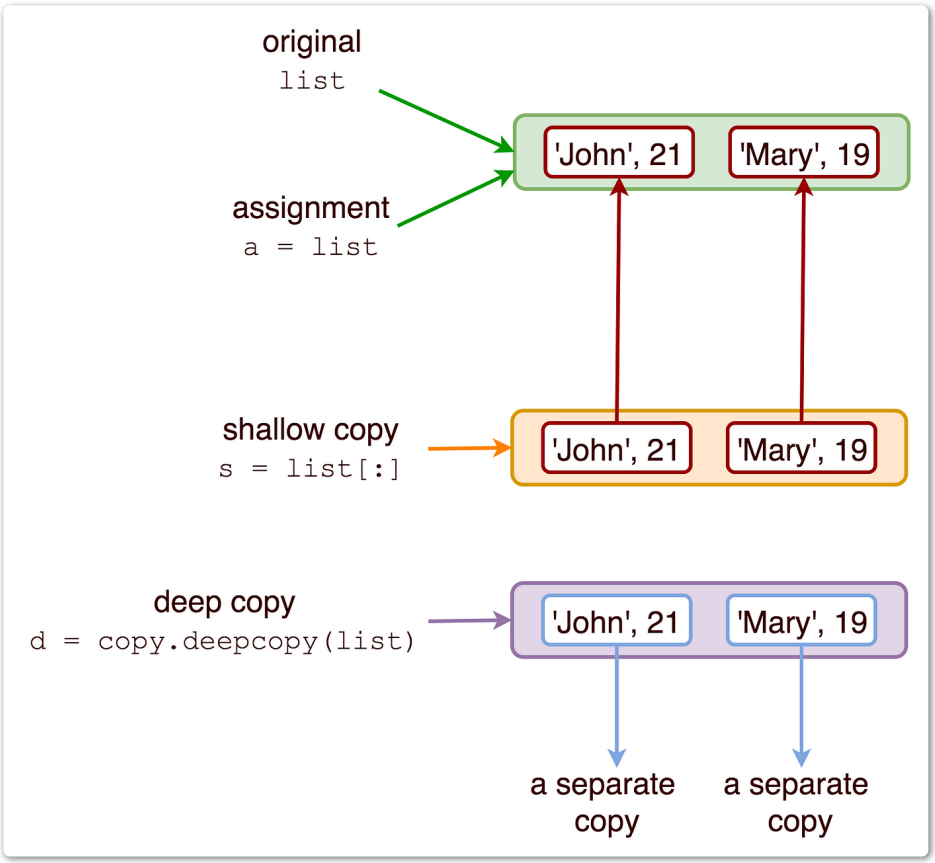
C a[2] = []

D a[2:3] = []

答案

1=>D　　2=>C

Python浅拷贝与深拷贝



- **直接赋值**：其实就是对象的引用（别名）。
- **浅拷贝(copy)**：拷贝父对象，不会拷贝对象的内部的子对象。
- **深拷贝(deepcopy)**： `copy` 模块的 `deepcopy` 方法，完全拷贝了父对象及其子对象。

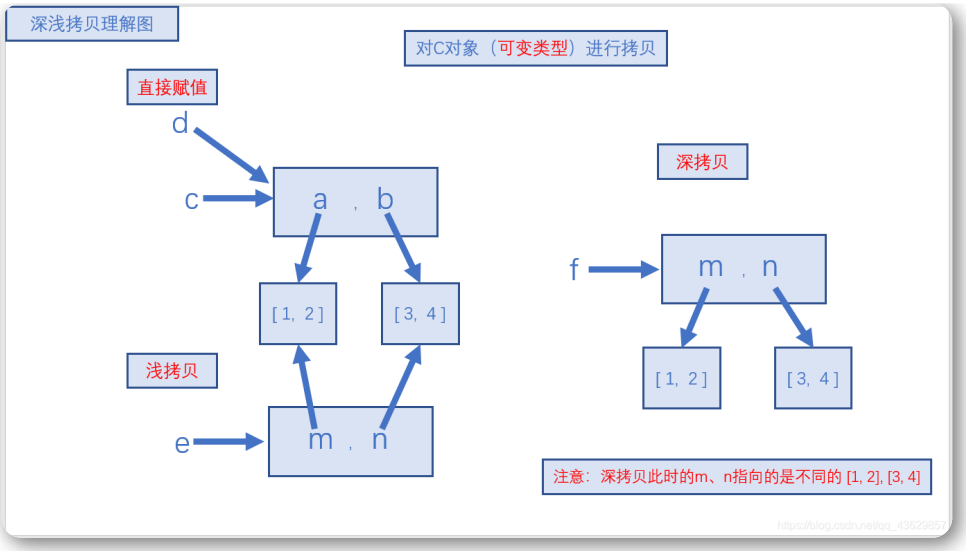
直接赋值代码示例，

```
a = [11, 22, 33]
b = a
print(b)
print(id(a), id(b))

c = {"name":"baizhan"}
d = c
print(id(c), id(d))
```

```
9
10 a.append(44)
11 print(a)
12 print(b)
13
14 c["age"] = 21
15 print(c)
16 print(d)
```

浅拷贝代码示例，

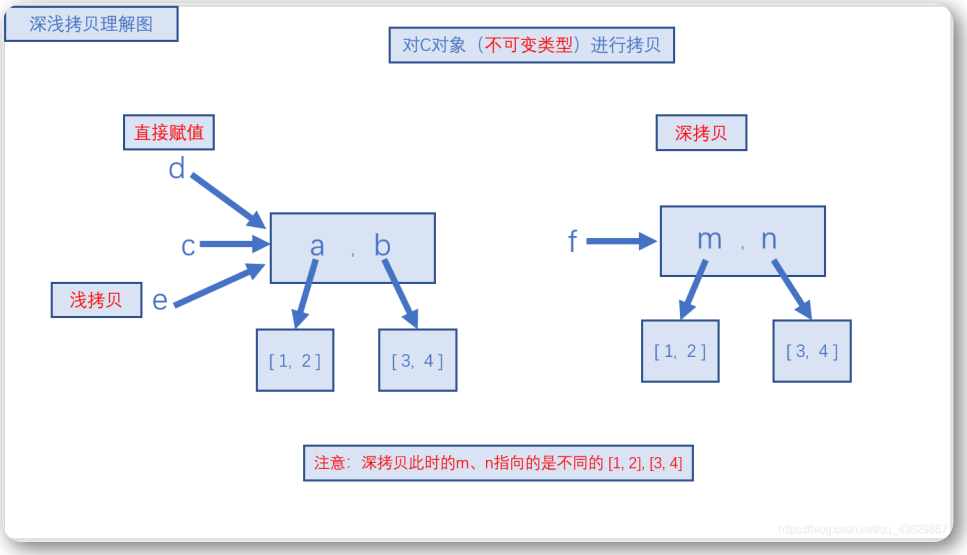
```
1  a = [1, 2]
2  b = [3, 4]
3  c = [a, b]
4  d = c
5  print(id(c), id(d))
6
7  import copy
8  e = c.copy()
9  print(id(c), id(d), id(e))
10 print(id(c[0]), id(c[1]))
11 print(id(e[0]), id(e[1]))
12
13 a.append(5)
14 b.append(6)
15 print(c)
16 print(d)
17 print(e)
```



深拷贝代码示例，

```
a = [1, 2]
b = [3, 4]
c = [a, b]
d = copy.deepcopy(c)
print(id(c), id(d))
print(id(c[0]), id(c[1]))
print(id(d[0]), id(d[1]))

c.append(5)
print(c)
print(d)

a.append(3)
b.append(5)
print(c)
print(d)

d[0].append(5)
d[1].append(6)
print(d)
d.append(7)
print(d)
```

```
print(c)
```



实时效果反馈

1. 下列代码进行的是赋值、浅拷贝还是深拷贝:

```
a = [1, 2]
b = [3, 4]
c = [a, b]
d = c.copy()
```

A 赋值

B 浅拷贝

C 深拷贝

D 以上都不是

2. 下面哪个说法是不正确的:

A 赋值: 只是复制了新对象的引用，不会开辟新的内存空间。

B 浅拷贝：重新分配一块内存，创建一个新的对象，但里面的元素是原对象中各个子对象的引用。

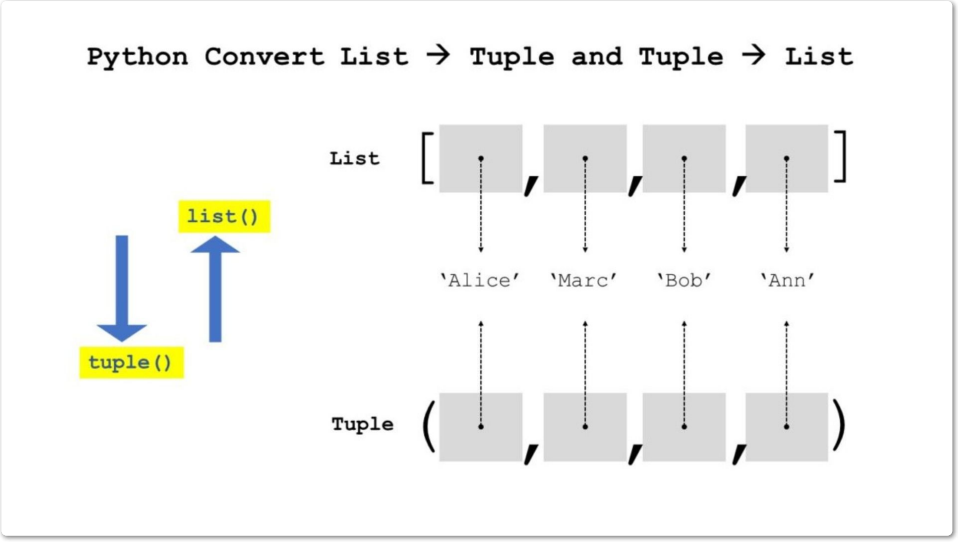
C 浅拷贝：不管值是可变对象还是不可变对象，拷贝前后的id值均不同

D 深拷贝：拷贝了对象的所有元素，包括多层嵌套的元素。深拷贝出来的对象是一个全新的对象，不再与原来的对象有任何关联。

答案

1=>B　　2=>C

Python元组



元组和列表一样，也是一种序列。不同的是元组不能修改，也就是说，元组是只读的，不能对元组进行增加、删除、修改。定义元组非常简单，只需要用逗号(，)分隔值即可。

元组的创建，通过()创建元组，小括号可以省略。

```
#通过()创建元组
a=1,2,3,4,5,6,7                #创建一个元组   省略了括号
b=(1,2,3,4,5,6,7)              #创建一个元组
c=(42,)                      #创建一个只有一个元素值的元组
d=()                         #创建一个空的元组
print(a)
print(b)
print(c)
print(d)
```

通过tuple()创建元组，

```
#使用tuple()函数创建元组
a=tuple("abc")
b=tuple(range(3))
c=tuple((1,2,3,4,5))
print(a)
print(b)
print(c)
```

元组不可修改

```
a[1]
a[1] = 20
del a[1]
```

元组的切片

```
a = (20,10,30,9,8)
print(a[1])
print(a[1:3])
print(a[:4])
```

列表关于排序的方法list.sort()是修改原列表对象，元组没有该方法。

如果要对元组排序，只能使用内置函数sorted(tupleObj)，并生成新的列表对象。

```
a = (20,10,30,9,8)
a.sort()
print(sorted(a))
```

> len()、max()、min()、sum()、all()、any()

序列解包

```
a,b,c = (10, 20, 30)
a,b,c = [10, 20, 30]
```

实时效果反馈

1. 判断下面代码是否是有效代码:

```
x = (1, 2, 3, 4)
del x
```

A 不是，因为元组是不可变的

B 是，会删除元组中的第一个元素

C 是，整个元组都会被删除

D 不是，对于del方法来说是无效语法

2. 下面代码的输出会是什么：

```
my_tuple=(2,3,4,5)
my_tuple.append((6,7,8))
print(len(my_tuple))
```

A 1

B 2

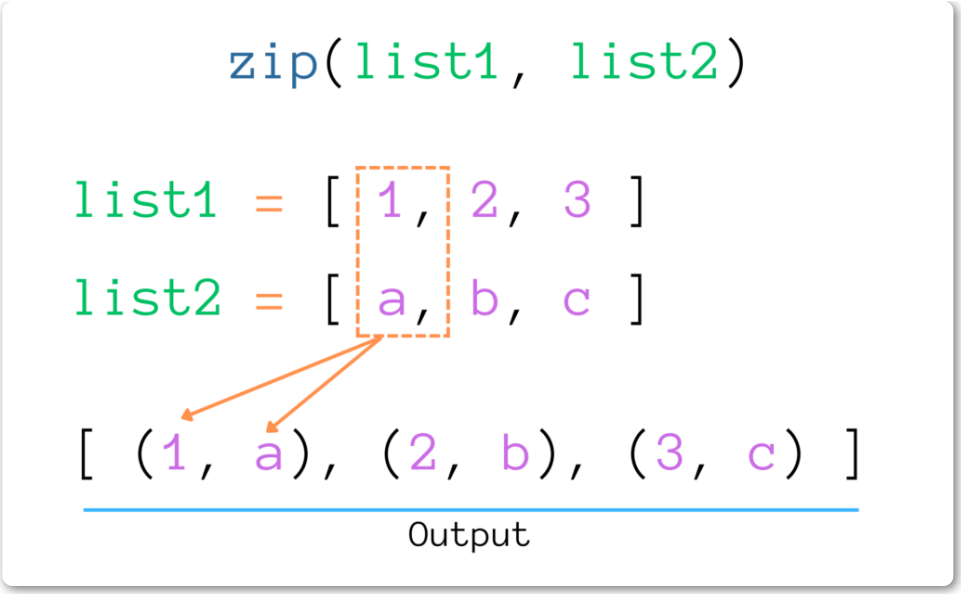
C 5

D 运行报错

答案

1=>C　　2=>D

Python Zip函数



zip(列表1，列表2，...)将多个列表对应位置的元素组合成为元组，并返回这个zip对象

```
a = [10, 20, 30]
b = [40, 50, 60]
zip(a,b)
tuple(zip(a,b))
```

```
a = [1, 2, 3]
b = ['我', '你', '他']
list(zip(a, b))
dict(zip(b, a))
```

upzip

```
a = [1, 2, 3]
b = ['我', '你', '他']
list(zip(a, b))

zipped_list = list(zip(a, b))
a, b = zip(*zipped_list)
```

可利用zip函数同时对多个长度相同的集合进行遍历

```
prog_langs = ["Python", "Java", "C",
"JavaScript"]
features = [10, 20, 30, 40]
for lang, feat in zip(prog_langs,features):
    print(lang, feat)
```

zip函数与sorted函数结合使用

```
prog_langs = ["Python", "Java", "C",
"JavaScript"]
features = [10, 20, 30, 40]
result =
sorted(list(zip(features,prog_langs)))
result
```

实时效果反馈

1. 下面代码的输出会是什么:

```
pairs=[(1,'a',True),(2,'b',False),
(3,'c',False),(4,'d',True)]
numbers,char,val= zip(*pairs)
print(numbers,char,val)
```

A (1,2,3,4) ('a', 'b', 'c', 'd') (True,False,False,True)

B 运行报错

C ((1,2,3,4),('a', 'b', 'c', 'd'),(True,False,False,True))

D 以上都不是

2. 下面代码的输出会是什么:

```
a = [1,2,3,4,5,6]
name = [“Jack”, “Smith”, “Paul”, “Brad”]
zipped =  zip(a, name)
print(list(zipped))
```

A [(1, ‘Jack’), (2, ‘Smith’), (3, ‘Paul’), (4, ‘Brad’), (5,), (6,)]

B [(1, ‘Jack’), (2, ‘Smith’), (3, ‘Paul’), (4, ‘Brad’)}

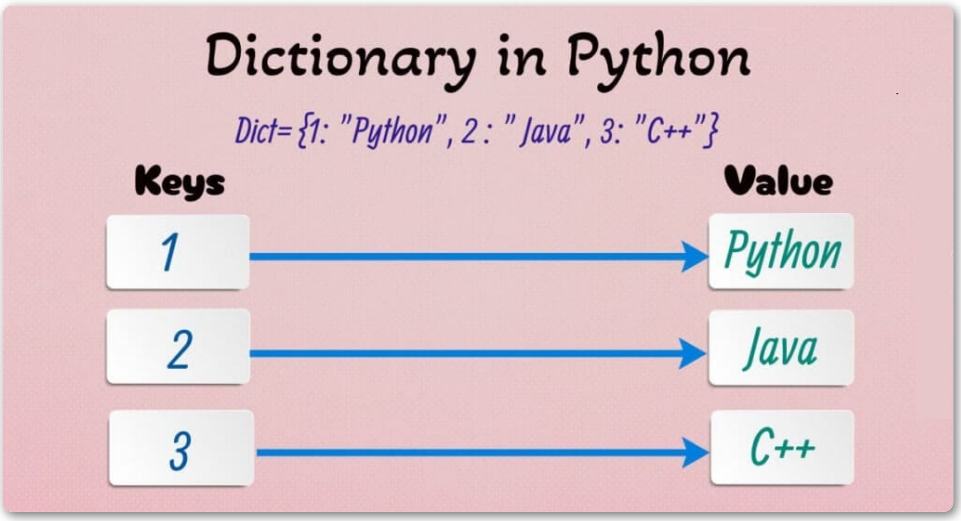
C 运行报错

D 以上都不是

答案

1=>A　　2=>B

Python字典



Python内置了字典(dict)的支持，dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度。字典是另一种可变容器模型，且可存储任意类型对象。字典的每个键值对(key=>value)用冒号(:)分割，每对之间用逗号(,)分割，整

个字典包括在花括号({})中，语法格式如下：

> d = {key1 : value1, key2 : value2 }

列表中通过“下标数字”找到对应的对象。字典中通过“键对象”找到对应的“值对象”。“键”是任意的不可变数据，比如：整数、浮点数、字符串、元组。但是：列表、字典、集合这些可变对象，不能作为“键”。并且“键”不可重复。“值”可以是任意的数据，并且可重复。

字典的创建

```
a = {'name':'baizhan','age':18,'job':'programmer'}
b = dict(name='baizhan',age=18,job='programmer')
c = dict([("name","baizhan"),("age",18),("job",'programmer')])
d = {}   #空的字典对象
e = dict()   #空的字典对象
print('字典a: ',a)
print('字典b: ',b)
print('字典c: ',c)
print('字典d: ',d)
print('字典e: ',e)
```

```
a = [1, 2, 3]
b = ['我', '你', '他']
list(zip(a, b))
dict(zip(b, a))
```

```
a = dict.fromkeys(['name','age','job'])
print('值为空的字典a:',a)
```

字典的访问

```
a =
{'name':'baizhan','age':18,'job':'programmer'
}
print('name:',a['name'])
print('age:',a['age'])
print('job:',a['job'])
```

get方法用于从字典中获取key对应的value。优点是：指定键不存在，返回None；也可以设定指定键不存在时默认返回的对象。推荐使用get()获取“值对象”。

```
a =
{'name':'baizhan','age':18,'job':'programmer'
}
print('name:',a.get('name'))
print('age:',a.get('age'))
print('job:',a.get('job'))
print('sex:',a.get('sex'))
print('sex:',a.get('sex', 0))
```

字典添加、修改元素操作

```
a = {'name':'baizhan','age':18,'job':'programmer'}
a['address']='北京'                #address的键不存在，则新增
a['age']=28                        #age的键存在，则进行修改
print(a)
```

del()、pop()

```
a = {'name':'baizhan','age':18,'job':'programmer'}
del(a['name'])          #del删除元素
print(a)
b=a.pop('age')          #pop删除元素，返回对应的值
print(b)
print(a)
a.clear()               #清空字典元素
print(a)
```

实时效果反馈

1. 下面说法不正确的是:

A 字典可以嵌到任意深度

B 字典是通过key来进行访问的

C 字典可以包含任意类型，但不能包含字典

D 字典是可变类型

2. 下面哪种方式不是正确定义字典的方式：

A

```
d = dict([
    ('foo', 100),
    ('bar', 200),
    ('baz', 300)
])
```

B d = {'foo': 100, 'bar': 200, 'baz': 300}

C d = dict(foo=100, bar=200, baz=300)

D

```
d = {
    ('foo', 100),
    ('bar', 200),
    ('baz', 300)
}
```

答案

1=>C　　2=>D

Python字典items()、keys()、values()、enumerate()

例子：将句子进行索引化

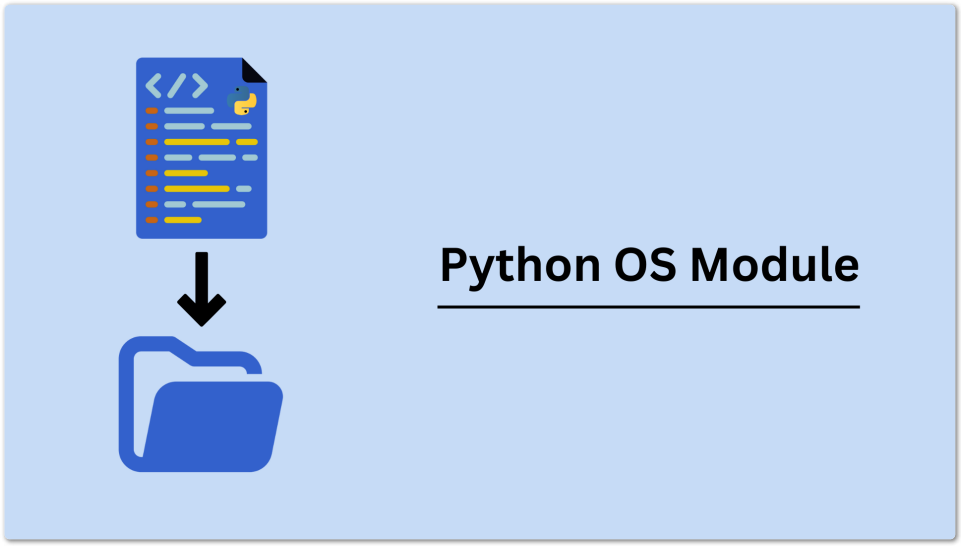
```
a = [1,2,3,4,5]
b = ['我', '你', '他', '爱', '天安门']
c = dict(zip(b,a))
c['我']

sentence = '我 爱 天安门'
words = sentence.split(' ')
sen_indexes = []
for word in words:
    sen_indexes.append(word)
    sen_indexes.append(c[word])
[c[word] for word in words]
# get方法
sentence = '我 爱 北京 天安门'
```

```
a =
{'name':'baizhan','age':18,'job':'programmer'
}
print(a.items())     #字典中所有的key-value对
print(a.keys())      #字典中所有的keys
print(a.values())    #字典中所有的value
#通过遍历key，根据key获取值
for key in a.keys():
    print(key,':',a.get(key))
```

enumerate()枚举函数

```
for i, key in enumerate(a):
    print(i, key, c[key])
```

Python之os模块



os模块以及子模块path中包含了大量操作文件和目录的函数

os.system可以直接调用系统的命令

```
import os
os.system("notepad.exe")
```

```
import os
os.system("ping www.baidu.com")
```

获取与改变工作目录

```
import os
#获取当前的工作目录，并输出该目录
print('当前工作目录：',os.getcwd())
#获取path指定的文件夹包含的文件或文件夹名称列表
print(os.listdir(os.getcwd()))
#改变当前工作目录
os.chdir('../')
print('改变后的工作目录：',os.getcwd())
print(os.listdir(os.getcwd()))
```

mkdir函数

```
import os
#使用mkdir创建目录
#判断当前目录下是否存在newfile目录
if not os.path.exists('newfile'):
    os.mkdir('newfile')
#mkdir函数不能创建多级目录，会抛出异常
os.mkdir('x/y/z')
```

对文件进行重命名

```
import os

if not os.path.exists('test'):
    os.mkdir('test')

d='./test'
os.listdir(d)

for file_name in os.listdir(d):
    print(file_name)
```

```
11      new_file_name =
file_name.replace('part', 'part-r-0000')
12      print(new_file_name)
13      os.path.join()
14      os.rename()
```

实时效果反馈

1. OS模块是一个___模块:

A 内置的（Built-in）

B 用户自定义的

C A和B

D 以上都不是

2. ___是OS模块的子模块:

A path

B sys

C math

D A和B

答案

1=>A　　2=>A

Python文件读写操作

打开文件

```
poem = '\n让我们一起学习AI\n'
f = open('./test/poem.txt','r')
f = open('./test/poem.txt','w')
f = open('./test/poem.txt','a')
f.write(poem)
f.close()
```

```
f = open('./test/poem.txt','r')
data = f.read()
print(data)
f.close()
```

```
with open('./test/poem.txt','r') as f:
    data = f.read()
    print(data)
```

readlines()

```
f = open('./test/poem.txt','r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

一行行的读

```
f = open('./test/poem.txt','r')
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()
```

编码格式，一般项目都会使用UTF-8

```
f = open('./test/poem.txt','r',encoding='utf-8')
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()
```

实时效果反馈

1. 哪一种方法被用于一行行读文件:

A read(1)

B readlines(1)

C readline()

D line()

2. 选择正确的方法将多行以一个列表写入文件：

A write(list)

B writelines(list)

C writelist(list)

D 以上都不是

答案

1=>C　　2=>B