1. Python标准库math中用来计算平方根的函数是\_\_\_\_\_\_\_\_\_\_。（sqrt()）
2. 在Python中\_\_\_\_\_\_\_\_\_\_表示空类型。（None）
3. 查看变量类型的Python内置函数是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（type（））
4. 查看变量内存地址的Python内置函数是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（id（））
5. Python运算符中用来计算整商的是\_\_\_\_\_\_\_\_\_。（//）

" / " 表示浮点数除法，返回浮点结果

" // " 表示整数除法,返回不大于结果的一个最大的整数

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_命令既可以删除列表中的一个元素，也可以删除整个列表。（del）

**remove**: 删除单个元素，删除首个符合条件的元素，按值删除

**pop**: 删除单个或多个元素，按位删除(根据索引删除)

**del**：它是根据索引(元素所在位置)来删除

除此之外，del还可以删除指定范围内的值。

del 也可以删除整个数据对象(列表、集合等)

1. 表达式 abs(-3) 的值为\_\_\_\_\_\_\_\_\_\_\_。（3）

abs() 函数返回数字的绝对值。

1. Python 3.x语句 print(1, 2, 3, sep=':') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_。（1:2:3）

分隔函数

1. 表达式 int(4\*\*0.5) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
2. Python内置函数\_\_\_\_\_\_\_\_\_可以返回列表、元组、字典、集合、字符串以及range对象中元素个数。（len()）

注意与count（）函数区分：统计字符串里某个字符出现的次数或指定元素在列表中出现的次数。

1. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_用来返回序列中的最大元素。（max()）
2. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_用来返回序列中的最小元素。（min()）
3. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_用来返回数值型序列中所有元素之和。（sum()）
4. 已知 x = 3，那么执行语句 x += 6 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（9）
5. 表达式type({3})的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（set）

Python3 中有六个标准的数据类型：

Number（数字）String（字符串）Tuple（元组）List（列表）Set（集合）Dictionary（字典）

1. 表达式[1, 2, 3]\*3的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3, 1, 2, 3, 1, 2, 3]）
2. list(map(str, [1, 2, 3]))的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[‘1’, ‘2’, ‘3’]）

str() 函数将对象转化为适于人阅读的形式。

**map()** 会根据提供的函数对指定序列做映射。

List变为列表

1. 表达式[3] in [1, 2, 3, 4]的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）

3 in [1, 2, 3, 4] 为True

1. 假设列表对象aList的值为[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]，那么切片aList[3:7]得到的值是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[6, 7, 9, 11]）
2. 假设有列表a = ['name', 'age', 'sex']和b = ['Dong', 38, 'Male']，请使用一个语句将这两个列表的内容转换为字典，并且以列表a中的元素为“键”，以列表b中的元素为“值”，这个语句可以写为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（c = dict(zip(a, b))）

**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。（为python2的版本）

**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（无法再返回列表）（为python3的版本）

1. 任意长度的Python列表、元组和字符串中最后一个元素的下标为\_\_\_\_\_\_\_\_。（-1）
2. Python语句list(range(1,10,3))执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 4, 7]）
3. 表达式 list(range(5)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 1, 2, 3, 4]）
4. 使用切片操作在列表对象x的开始处增加一个元素3的代码为\_\_\_\_\_\_\_\_\_。（x[0:0] = [3]）
5. 语句 x = (3,) 执行后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（(3,)）
6. 语句 x = (3) 执行后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）
7. 已知x=3和y=5，执行语句 x, y = y, x 后x的值是\_\_\_\_。（5）
8. 已知 x = {1:2}，那么执行语句 x[2] = 3之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1: 2, 2: 3}）

没有键2会自动添加

1. 表达式set([1, 1, 2, 3])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3}）
2. 使用列表推导式得到100以内所有能被13整除的数的代码可以写作\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[i for i in range(100) if i%13==0]） （注意range（100）的括号）
3. 已知 x = [3, 5, 7]，那么表达式 x[10:]的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[]）
4. 已知 x = [3, 7, 5]，那么执行语句 x.sort(reverse=True)之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[7, 5, 3]）   
   先排序，然后降序
5. 表达式 list(zip([1,2], [3,4])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[(1, 3), (2, 4)]）
6. 已知 x = [1, 2, 3, 2, 3]，执行语句 x.pop() 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3, 2]）

默认删除最后一个元素

1. 表达式 [x for x in [1,2,3,4,5] if x<3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2]）
2. 表达式 [index for index, value in enumerate([3,5,7,3,7]) if value == max([3,5,7,3,7])] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[2, 4]）

返回最大值所在的下标

1. 已知 x = list(range(10))，则表达式 x[-4:] 的值为\_\_\_\_\_\_\_\_\_\_。（[6, 7, 8, 9]）

负数索引，从最后开始往前数，最后为-1

1. 已知 x = [3, 5, 7]，那么执行语句 x[1:] = [2]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[3, 2]）
2. 已知 x = [3, 5, 7]，那么执行语句 x[:3] = [2]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[2]）
3. 已知 x = [1, 2, 3, 2, 3]，执行语句 x.remove(2) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 3, 2, 3]）

删除首次出现的指定元素，不是下标

1. 表达式 len([i for i in range(10)]) 的值为\_\_\_\_\_\_\_\_\_\_。（10）
2. 已知 x = (3,)，那么表达式 x \* 3 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（(3, 3, 3)）
3. 已知列表 x = list(range(10))，那么执行语句 del x[::2]之后，x的值为\_\_\_\_\_\_\_\_\_。（[1, 3, 5, 7, 9]）
4. 已知列表 x = [1, 2, 3, 4]，那么执行语句 del x[1] 之后x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 3, 4]）
5. 已知列表 x = [1, 2, 3]，那么执行语句 x.insert(1, 4) 只有，x的值为\_\_\_\_\_\_\_\_\_\_\_。（[1, 4, 2, 3]）
6. 表达式 list(map(lambda x: x+5, [1, 2, 3, 4, 5])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[6, 7, 8, 9, 10]）
7. 已知 x = [1, 2, 3]，那么执行语句 x[len(x)-1:] = [4, 5, 6]之后，变量x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 4, 5, 6]）
8. 已知x是一个列表对象，那么执行语句 y = x[:] 之后表达式 id(x) == id(y) 的值为\_\_\_\_\_\_\_\_\_\_\_。（False）

切片为浅复制，是副本，储存单位不同（与XXX=XXX区分）

1. 表达式 sorted([13, 1, 237, 89, 100], key=lambda x: len(str(x))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 13, 89, 237, 100]）

Key参数是按照自己设置的方式进行排序，这里按照数字长度进行排序

1. 已知 x = {1:2, 2:3}，那么表达式 x.get(3, 4) 的值为\_\_\_\_。（4）

Get方法对于不存在且无指定值的键返回none，不存在但指定指定值的返回指定值。

1. 已知 x = {1:2, 2:3}，那么表达式 x.get(2, 4) 的值为\_\_\_\_\_\_。（3）
2. 表达式 [1,2,3].count(4) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
3. 执行代码 x, y, z = sorted([1, 3, 2]) 之后，变量y的值为\_\_\_\_\_\_。（2）
4. 表达式 dict(zip([1, 2], [3, 4])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1: 3, 2: 4}）
5. 已知 x = {1:1}，那么执行语句 x[2] = 2之后，len(x)的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
6. 已知 x = {1:1, 2:2}，那么执行语句 x[2] = 4之后，len(x)的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
7. 已知有函数定义 def demo(\*p):return sum(p)，那么表达式 demo(1, 2, 3) 的值为\_\_\_\_\_\_、表达式 demo(1, 2, 3, 4) 的值为\_\_\_\_\_\_\_\_\_。（6、10）
8. 已知列表 x = [1, 2]，那么连续执行命令 y = x和 y.append(3) 之后，x的值为\_\_\_\_\_\_\_\_。（[1, 2, 3]）

X与y的地址相同，改编y的值，x的值也发生变化

1. 已知列表 x = [1, 2]，那么连续执行命令 y = x[:] 和 y.append(3) 之后，x的值为\_\_\_\_\_\_\_\_。（[1, 2]）
2. 已知列表 x = [1, 2]，执行语句 y = x[:] 后，表达式 id(x) == id(y) 的值为\_\_\_\_\_\_\_。（False）
3. 已知列表 x = [1, 2]，执行语句 y = x 后，表达式 id(x) == id(y) 的值为\_\_\_\_\_\_\_。（True）
4. 已知列表 x = [1, 2]，执行语句 y = x 后，表达式 x is y 的值为\_\_\_\_\_\_\_。（True）
5. 已知列表 x = [1, 2]，执行语句 y = x[:] 后，表达式 x is not y 的值为\_\_\_\_\_\_\_。（True）

is 是判断两个标识符是不是引用自一个对象，**x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False

is not 是判断两个标识符是不是引用自不同对象，**x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。

1. 表达式 [i for i in range(10) if i>8] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[9]）
2. 执行语句 x,y,z = map(str, range(3)) 之后，变量y的值为\_\_\_\_\_\_\_\_\_\_\_。（‘1’）
3. 已知 x = [1, 2, 3, 4, 5]，那么执行语句 del x[:3] 之后，x的值为\_\_\_\_\_\_\_\_\_\_。（[4, 5]）
4. 已知 x = [1, 2]，那么执行语句 x[0:0] = [3, 3]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_。（[3, 3, 1, 2]）
5. 已知 x = [1, 2, 3, 4, 5]，那么执行语句 del x[1:3] 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 4, 5]）
6. 已知 x = {1:2, 2:3, 3:4}，那么表达式 sum(x.values()) 的值为\_\_\_\_\_\_\_\_\_。（9）

Value：返回键值

1. 已知列表 x = [1, 3, 2]，那么执行语句 x.reverse() 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[2, 3, 1]）
2. 已知列表 x = [1, 3, 2]，那么执行语句 y = list(reversed(x)) 之后，x的值为\_\_\_\_\_\_\_\_\_\_。（[1, 3, 2]）
3. 已知列表 x = [1, 3, 2]，那么执行语句 y = list(reversed(x)) 之后，y的值为\_\_\_\_\_\_\_\_\_\_。[2,3,1]
4. 已知字典 x = {i:str(i+3) for i in range(3)}，那么表达式 sum(x) 的值为\_\_\_\_\_\_。 3
5. 已知字典 x = {i:str(i+3) for i in range(3)}，那么表达式 sum(item[0] for item in x.items()) 的值为\_\_\_\_\_\_\_\_\_\_\_。 3

Python 字典(Dictionary) items() 函数以列表返回可遍历的(键, 值) 元组数组

1. 已知列表 x = [1, 3, 2]，那么执行语句 a, b, c = sorted(x) 之后，b的值为\_\_\_\_。2
2. 已知列表 x = [1, 3, 2]，那么执行语句 a, b, c = map(str,sorted(x)) 之后，c的值为\_\_\_\_\_\_。（‘3’）
3. 表达式 'abc' in 'abdcefg' 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。 False
4. 已知 x, y = 3, 5，那么执行x, y = y, x 之后，x的值为\_\_\_\_\_\_\_\_\_\_。5
5. 已知 x = 'abcd' 和 y = 'abcde'，那么表达式 [i==j for i,j in zip(x,y)] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [True, True, True, True]

zip()函数接受一系列可迭代对象作为参数，将不同对象中相对应的元素打包成一个元组（tuple），返回由这些元组组成的list列表，如果传入的参数的长度不等，则返回的list列表的长度和传入参数中最短对象的长度相同。

1. 已知x = list(range(20))，那么表达式x[-1]的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 19
2. 已知x = 3+4j和y = 5+6j，那么表达式x+y的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 (8+10j)
3. 已知x = [3]，那么执行x += [5]之后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [3,5]
4. 已知x = [3, 2, 4, 1]，那么执行语句x = x.sort()之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。 [1,2,3,4]
5. 表达式list(filter(lambda x: x>5, range(10)))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [6,7,8,9]
6. 已知x = list(range(20))，那么语句print(x[100:200])的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。[]
7. 已知x = list(range(20))，那么执行语句x[:18] = []后列表x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [18,19]
8. 已知x = [1, 2, 3]，那么连续执行y = x[:]和y.append(4)这两条语句之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [1,2,3]
9. 已知x = [1, 2, 3]，那么连续执行y = x和y.append(4)这两条语句之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [1,2,3,4]
10. 已知x = [1, 2, 3]，那么连续执行y = [1, 2, 3]和y.append(4)这两条语句之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 [1,2,3]
11. 已知x = ([1], [2])，那么执行语句x[0].append(3)后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ([1, 3], [2])
12. 表达式 True\*3 的值为\_\_\_\_\_\_\_\_\_\_。 3
13. 表达式 False+1 的值为\_\_\_\_\_\_\_\_\_\_。 1
14. 表达式 'ab' in 'acbed' 的值为\_\_\_\_\_\_\_\_。 False
15. Python 3.x语句 for i in range(3):print(i, end=',') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。0,1,2，
16. Python 3.x语句 print(1, 2, 3, sep=',') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。1,2,3
17. 对于带有else子句的for循环和while循环，当循环因循环条件不成立而自然结束时\_\_\_\_\_\_\_\_(会？不会？)执行else中的代码。 会
18. 在循环语句中，\_\_\_\_\_\_\_\_\_\_语句的作用是提前结束本层循环。 break
19. 在循环语句中，\_\_\_\_\_\_\_语句的作用是提前进入下一次循环。 continue
20. 表达式 5 if 5>6 else (6 if 3>2 else 5) 的值为\_\_\_\_\_\_\_\_\_。 6
21. 表达式 'abc' in ('abcdefg') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 True
22. 表达式 'abc' in ['abcdefg'] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 False
23. Python语句''.join(list('hello world!'))执行的结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘hello world!’
24. 表达式 'abcabcabc'.rindex('abc') 的值为\_\_\_\_\_\_\_\_\_\_\_\_。 6
25. 表达式 ':'.join('abcdefg'.split('cd')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 （’ab:efg’）
26. 表达式 'Hello world. I like Python.'.rfind('python') 的值为\_\_\_\_\_\_\_\_。 -1 大小写
27. 表达式 'abcabcabc'.count('abc') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。 3
28. 表达式 'apple.peach,banana,pear'.find('p') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 1

返回查找到的第一个字符

1. 表达式 'apple.peach,banana,pear'.find('ppp') 的值为\_\_\_\_\_\_\_\_。 -1
2. 表达式 'abcdefg'.split('d') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ['abc', 'efg']
3. 表达式 ':'.join('1,2,3,4,5'.split(',')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘1:2:3:4:5’
4. 已知 x = '123' 和 y = '456'，那么表达式 x + y 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（’123456’）
5. 表达式 ''.join('asdssfff'.split('sd')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。 ‘assfff’
6. 表达式 ''.join(re.split('[sd]','asdssfff')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。’afff’
7. 假设re模块已导入，那么表达式 re.findall('(\d)\\1+', '33abcd112') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ['3', '1']
8. 语句 print(re.match('abc', 'defg')) 输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_。 None

re.match(pattern, string, flags)  
第一个参数是正则表达式,如果匹配成功，则返回一个Match，否则返回一个None；  
第二个参数表示要匹配的字符串；  
第三个参数是标致位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等

1. 表达式 'Hello world!'[-4] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘r’
2. 表达式 'Hello world!'[-4:] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘rld!’
3. 表达式 len('abc'.ljust(20)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 20

**字符串在输出时的对齐:**

**S.ljust(width,[fillchar])**

**#输出width个字符，S左对齐，不足部分用fillchar填充，默认的为空格。**

**S.rjust(width,[fillchar]) #右对齐**

**S.center(width, [fillchar]) #中间对齐**

**S.zfill(width) #把S变成width长，并在右对齐，不足部分用0补足**

1. 假设正则表达式模块re已导入，那么表达式 re.sub('\d+', '1', 'a12345bbbb67c890d0e') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘a1bbbb1c1d1e’ re.sub用于替换字符串中的匹配项
2. 表达式 [str(i) for i in range(3)] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。 ['0', '1', '2']
3. 表达式 'abc.txt'.endswith(('.txt', '.doc', '.jpg')) 的值为\_\_\_\_\_\_\_\_\_\_\_。 True

**str.endswith("suffix", start, end) 判断字符串是否以指定字符或子字符串结尾。**

1. 代码 print(1,2,3,sep=':') 的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 1:2:3

表示当输出多个打印的值时，各个值之间分割方式， 默认空格，可以自定义

1. 代码 for i in range(3):print(i, end=',') 的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 0,1,2,
2. 表达式 'Hello world!'.count('l') 的值为\_\_\_\_\_\_\_\_\_\_\_。 3
3. 已知 x = 'abcdefg'，则表达式 x[3:] + x[:3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘defgabc’
4. 已知x = 'hello world.'，那么表达式x.find('x')和x.rfind('x')的值都为\_\_\_\_\_\_\_\_\_\_\_\_\_。-1

**find**是从字符串**左边**开始查询,查询到的第一个子字符串的下标(**下标从0开始**)  
**rfind**是从字符串**右边**开始查询,查询到的第一个子字符串的下标(**下标从0开始**)

1. 表达式':'.join('hello world.'.split())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘hello:world.’

单引号框起来的是分隔符

1. 已知x = 'hello world'，那么表达式x.replace('l', 'g')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘heggo worgd’
2. 假设正则表达式模块re已正确导入，那么表达式''.join(re.findall('\d+', 'abcd1234'))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ‘1234’

Python的join()方法用于将序列中的元素以指定的字符连接生成一个新的字符串

1. 假设正则表达式模块re已正确导入，那么表达式re.findall('\d+?', 'abcd1234')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。 ['1', '2', '3', '4']
2. 正则表达式模块re的\_\_\_\_\_\_\_\_\_\_方法用来编译正则表达式对象。 compile()
3. 正则表达式模块re的\_\_\_\_\_\_match()\_\_\_\_\_\_\_\_方法用来在字符串开始处进行指定模式的匹配.
4. 正则表达式模块re的\_\_\_\_\_\_\_search()\_\_\_\_\_\_\_方法用来在整个字符串中进行指定模式的匹配。 search返回找到的第一个 findall将匹配的所有字符串做列表返回

都要加括号

1. 表达式 'Beautiful is better than ugly.'.startswith('Be', 5) 的值为\_\_\_\_\_\_\_\_\_。 False

Python startswith() 方法用于检查字符串是否是以指定子字符串开头，如果是则返回 True，否则返回 False。如果参数 beg 和 end 指定值，则在指定范围内检查

1. 表达式 list(filter(lambda x:x>2, [0,1,2,3,0,0])) 的值为\_\_\_\_\_\_\_\_\_。 [3]
2. 表达式 list(filter(lambda x: len(x)>3, ['a', 'b', 'abcd'])) 的值为\_\_\_\_\_\_\_\_\_\_\_。 [‘abcd’]

filter()函数接收两个参数，第一个为函数，第二个为可迭代对象。返回序列中能使函数为真的元素的迭代器对象；如果函数为None，则返回序列中值为True的元素。

1. 已知 g = lambda x, y=3, z=5: x\*y\*z，则语句 print(g(1)) 的输出结果为\_\_\_\_15\_\_\_。
2. 表达式 list(map(lambda x: len(x), ['a', 'bb', 'ccc'])) 的值为\_\_\_\_\_[1,2,3]\_\_\_\_\_。

**map(func,list)函数有两个参数，前面一个是函数，后面一个是序列。该函数的意义是对一个序列进行前面参数函数的操作，然后返回一个新的list。**

1. 已知 f = lambda x: x+5，那么表达式 f(3) 的值为\_\_\_\_8\_\_\_\_。
2. 已知 g = lambda x, y=3, z=5: x+y+z，那么表达式 g(2) 的值为\_\_\_\_10\_\_\_\_。

输入值参数赋值给x

1. 已知 f = lambda x: 5，那么表达式 f(3)的值为\_\_\_\_\_\_5\_\_\_\_\_\_\_。
2. Python使用\_\_\_\_\_\_\_\_\_class\_\_\_\_\_\_\_\_关键字来定义类。