《Python程序设计》题库

1. 填空题

第一章 基础知识

1. Python安装扩展库常用的是\_\_\_\_\_\_\_工具。（pip）
2. Python标准库math中用来计算平方根的函数是\_\_\_\_\_\_\_\_\_\_。（sqrt）
3. Python程序文件扩展名主要有\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_两种，其中后者常用于GUI程序。（py、pyw）
4. Python源代码程序编译后的文件扩展名为\_\_\_\_\_\_\_\_\_。（pyc）
5. 使用pip工具升级科学计算扩展库numpy的完整命令是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（pip install --upgrade numpy）
6. 使用pip工具查看当前已安装的Python扩展库的完整命令是\_\_\_\_\_\_\_\_\_\_\_\_\_。（pip list）
7. 在IDLE交互模式中浏览上一条语句的快捷键是\_\_\_\_\_\_\_\_\_\_。（Alt+P）
8. 在Python中\_\_\_\_\_\_\_\_\_\_表示空类型。（None）
9. 列表、元组、字符串是Python的\_\_\_\_\_\_\_\_\_(有序？无序)序列。（有序）
10. 查看变量类型的Python内置函数是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（type()）
11. 查看变量内存地址的Python内置函数是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（id()）
12. 以3为实部4为虚部，Python复数的表达形式为\_\_\_\_\_\_\_\_\_\_\_或\_\_\_\_\_\_\_\_。（3+4j、3+4J）
13. Python运算符中用来计算整商的是\_\_\_\_\_\_\_\_\_。（//）
14. Python运算符中用来计算集合并集的是\_\_\_\_\_\_\_。（|）
15. 使用运算符测试集合包含集合A是否为集合B的真子集的表达式可以写作\_\_\_\_\_\_\_。（A<B）
16. \_\_\_\_\_\_\_\_\_\_\_\_\_\_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）
17. 表达式 int('123', 16) 的值为\_\_\_\_\_\_\_\_\_。（291）
18. 表达式 int('123', 8) 的值为\_\_\_\_\_\_\_\_\_。（83）
19. 表达式 int('123') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（123）
20. 表达式 int('101',2) 的值为\_\_\_\_\_\_\_\_\_\_。（5）
21. 表达式 abs(-3) 的值为\_\_\_\_\_\_\_\_\_\_\_。（3）
22. Python 3.x语句 print(1, 2, 3, sep=':') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_。（1:2:3）
23. 表达式 int(4\*\*0.5) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
24. 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()）
25. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_用来返回序列中的最大元素。（max()）
26. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_用来返回序列中的最小元素。（min()）
27. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_用来返回数值型序列中所有元素之和。（sum()）
28. 为了提高Python代码运行速度和进行适当的保密，可以将Python程序文件编译为扩展名\_\_\_\_\_\_\_\_\_\_的文件。（pyc）
29. 已知 x = 3，那么执行语句 x += 6 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（9）
30. 表达式 3 | 5 的值为\_\_\_\_\_\_\_\_\_\_。（7）
31. 表达式 3 & 6 的值为\_\_\_\_\_\_\_\_\_。（2）
32. 表达式 3 \*\* 2 的值为\_\_\_\_\_\_\_\_\_。（9）
33. 表达式 3 \* 2的值为\_\_\_\_\_\_\_\_\_\_\_。（6）
34. 表达式 3<<2 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（12）
35. 表达式 65 >> 1 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（32）
36. 表达式 chr(ord('a')^32) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'A'）
37. 表达式 chr(ord('a')-32) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'A'）
38. 表达式 abs(3+4j) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（5.0）
39. 表达式 callable(int) 的值为\_\_\_\_\_\_\_\_\_\_\_。（True）
40. 假设有Python程序文件abc.py，其中只有一条语句print(\_\_name\_\_)，那么直接运行该程序时得到的结果为\_\_\_\_\_\_\_\_\_\_\_\_\_。（\_\_main\_\_）
41. 表达式16\*\*0.5的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（4.0）
42. 表达式type({3})的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（set）
43. 表达式isinstance('Hello world', str)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）

**第二章 Python序列**

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’]）
3. 语句x = 3==3, 5执行结束后，变量x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（(True, 5））
4. 已知 x = 3，并且id(x)的返回值为 496103280，那么执行语句 x += 6 之后，表达式 id(x) == 496103280 的值为\_\_\_\_\_\_\_\_\_\_\_。（False）
5. 已知 x = 3，那么执行语句 x \*= 6 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（18）
6. 表达式[3] in [1, 2, 3, 4]的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
7. 列表对象的sort()方法用来对列表元素进行原地排序，该函数返回值为 。（None）
8. 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]）
9. 使用列表推导式生成包含10个数字5的列表，语句可以写为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[5 for i in range(10)]）
10. 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))）
11. 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）
12. Python语句list(range(1,10,3))执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 4, 7]）
13. 表达式 list(range(5)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 1, 2, 3, 4]）
14. 已知a = [1, 2, 3]和b = [1, 2, 4]，那么id(a[1])==id(b[1])的执行结果为\_\_\_\_\_\_\_\_\_\_\_。（True）
15. 切片操作list(range(6))[::2]执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 2, 4]）
16. 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]）
17. 语句sorted([1, 2, 3], reverse=True) == reversed([1, 2, 3])执行结果为\_\_\_\_\_\_\_\_\_。（False）
18. 表达式 sorted([111, 2, 33], key=lambda x: len(str(x))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[2, 33, 111]）
19. 表达式 sorted([111, 2, 33], key=lambda x: -len(str(x))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[111, 33, 2]）
20. 语句 x = (3,) 执行后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（(3,)）
21. 语句 x = (3) 执行后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）
22. 已知x=3和y=5，执行语句 x, y = y, x 后x的值是\_\_\_\_。（5）
23. 可以使用内置函数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_查看包含当前作用域内所有全局变量和值的字典。（globals()）
24. 可以使用内置函数\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_查看包含当前作用域内所有局部变量和值的字典。（locals()(）
25. 字典中多个元素之间使用\_\_\_\_\_\_\_\_\_\_\_\_分隔开，每个元素的“键”与“值”之间使用\_\_\_\_\_\_\_\_分隔开。（逗号、冒号）
26. 字典对象的\_\_\_\_\_\_\_\_\_\_\_方法可以获取指定“键”对应的“值”，并且可以在指定“键”不存在的时候返回指定值，如果不指定则返回None。（get()）
27. 字典对象的\_\_\_\_\_\_\_\_\_\_\_\_\_方法返回字典中的“键-值对”列表。（items()）
28. 字典对象的\_\_\_\_\_\_\_\_\_\_\_\_方法返回字典的“键”列表。（keys()）
29. 字典对象的\_\_\_\_\_\_\_\_\_\_\_\_\_\_方法返回字典的“值”列表。（values()）
30. 已知 x = {1:2}，那么执行语句 x[2] = 3之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1: 2, 2: 3}）
31. 表达式 {1, 2, 3, 4} - {3, 4, 5, 6}的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2}）
32. 表达式set([1, 1, 2, 3])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3}）
33. 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]）
34. 已知 x = [3, 5, 7]，那么表达式 x[10:]的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[]）
35. 已知 x = [3, 5, 7]，那么执行语句 x[len(x):] = [1, 2]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[3, 5, 7, 1, 2]）
36. 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]）
37. 已知 x = [3, 7, 5]，那么执行语句 x = x.sort(reverse=True)之后，x的值为\_\_\_\_\_\_\_\_\_。（None）
38. 已知 x = [1, 11, 111]，那么执行语句 x.sort(key=lambda x: len(str(x)), reverse=True) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[111, 11, 1]）
39. 表达式 list(zip([1,2], [3,4])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[(1, 3), (2, 4)]）
40. 已知 x = [1, 2, 3, 2, 3]，执行语句 x.pop() 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3, 2]）
41. 表达式 list(map(list,zip(\*[[1, 2, 3], [4, 5, 6]]))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1, 4], [2, 5], [3, 6]]）
42. 表达式 [x for x in [1,2,3,4,5] if x<3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2]）
43. 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]）
44. 已知 x = [3,5,3,7]，那么表达式 [x.index(i) for i in x if i==3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 0]）
45. 已知列表 x = [1, 2]，那么表达式 list(enumerate(x)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[(0, 1), (1, 2)]）
46. 已知 vec = [[1,2], [3,4]]，则表达式 [col for row in vec for col in row] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3, 4]）
47. 已知 vec = [[1,2], [3,4]]，则表达式 [[row[i] for row in vec] for i in range(len(vec[0]))] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1, 3], [2, 4]]）
48. 已知 x = list(range(10))，则表达式 x[-4:] 的值为\_\_\_\_\_\_\_\_\_\_。（[6, 7, 8, 9]）
49. 已知 x = [3, 5, 7]，那么执行语句 x[1:] = [2]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[3, 2]）
50. 已知 x = [3, 5, 7]，那么执行语句 x[:3] = [2]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[ 2]）
51. 已知x为非空列表，那么执行语句y = x[:]之后，id(x[0]) == id(y[0])的值为\_\_\_\_\_\_\_\_\_\_。（True）
52. 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]）
53. 表达式 len([i for i in range(10)]) 的值为\_\_\_\_\_\_\_\_\_\_。（10）
54. 表达式 len(range(1,10)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（9）
55. 表达式 range(10)[-1] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（9）
56. 表达式 range(10,20)[4] 的值为\_\_\_\_\_\_\_\_\_\_。（14）
57. 表达式 round(3.4) 的值为\_\_\_\_\_\_\_\_\_\_\_。（3）
58. 表达式 round(3.7) 的值为\_\_\_\_\_\_\_\_\_。（4）
59. 已知 x = (3)， 那么表达式 x \* 3 的值为\_\_\_\_\_\_\_\_\_\_。（9）
60. 已知 x = (3,)，那么表达式 x \* 3 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（(3, 3, 3)）
61. 假设列表对象x = [1, 1, 1]，那么表达式id(x[0]) == id(x[2])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
62. 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]）
63. 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]）
64. 表达式 [1] \* 2 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 1]）
65. 表达式 [1, 2] \* 2 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 1, 2]）
66. 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]）
67. 已知列表 x = [1, 2, 3]，那么执行语句 x.insert(0, 4) 只有，x的值为\_\_\_\_\_\_\_\_\_\_\_。（[4, 1, 2, 3]）
68. 已知列表 x = [1, 2, 3]，那么执行语句 x.pop(0) 之后，x的值为\_\_\_\_\_\_\_\_\_。（[2, 3]）
69. 已知 x = [[1]] \* 3，那么执行语句 x[0][0] = 5之后，变量x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[5], [5], [5]]）
70. 表达式 list(map(lambda x: x+5, [1, 2, 3, 4, 5])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[6, 7, 8, 9, 10]）
71. 表达式 {1, 2, 3, 4, 5} ^ {4, 5, 6, 7} 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3, 6, 7}）
72. 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]）
73. 表达式 len(range(1, 10)) 的值为\_\_\_\_\_\_\_\_\_\_\_。（9）
74. 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）
75. 表达式 sorted([13, 1, 237, 89, 100], key=lambda x: len(str(x))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 13, 89, 237, 100]）
76. 已知 x = {1:2, 2:3}，那么表达式 x.get(3, 4) 的值为\_\_\_\_。（4）
77. 已知 x = {1:2, 2:3}，那么表达式 x.get(2, 4) 的值为\_\_\_\_\_\_。（3）
78. 表达式 {1, 2, 3} | {3, 4, 5} 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3, 4, 5}）
79. 表达式 {1, 2, 3} | {2, 3, 4} 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3, 4}）
80. 表达式 {1, 2, 3} & {3, 4, 5} 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{3}）
81. 表达式 {1, 2, 3} & {2, 3, 4} 的值为\_\_\_\_\_\_\_\_\_\_\_。（{2, 3}）
82. 表达式 {1, 2, 3} - {3, 4, 5} 的值为\_\_\_\_\_\_\_\_\_。（{1, 2}）
83. 表达式 {1, 2, 3} < {3, 4, 5} 的值为\_\_\_\_\_\_\_\_\_。（False）
84. 表达式 {1, 2, 3} < {1, 2, 4} 的值为\_\_\_\_\_\_\_\_\_\_\_。（False）
85. 表达式 [1,2,3].count(4) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
86. Python标准库random中的\_\_\_\_\_\_\_\_\_\_\_方法作用是从序列中随机选择1个元素。（choice()）
87. Python标准库random中的sample(seq, k)方法作用是从序列中选择\_\_\_\_\_\_\_\_(重复？不重复？)的k个元素。（不重复）
88. random模块中\_\_\_\_\_\_\_\_\_\_\_\_\_方法的作用是将列表中的元素随机乱序。（shuffle()）
89. 执行代码 x, y, z = sorted([1, 3, 2]) 之后，变量y的值为\_\_\_\_\_\_。（2）
90. 表达式 (1, 2, 3)+(4, 5) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（(1, 2, 3, 4, 5)）
91. 表达式 dict(zip([1, 2], [3, 4])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1: 3, 2: 4}）
92. 语句 x, y, z = [1, 2, 3] 执行后，变量y的值为\_\_\_\_\_\_\_\_\_\_。（2）
93. 已知 x = [[1,3,3], [2,3,1]]，那么表达式 sorted(x, key=lambda item:item[0]+item[2]) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[2, 3, 1], [1, 3, 3]]）
94. 已知 x = [[1,3,3], [2,3,1]]，那么表达式 sorted(x, key=lambda item:(item[1],item[2])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[2, 3, 1], [1, 3, 3]]）
95. 已知 x = [[1,3,3], [2,3,1]]，那么表达式 sorted(x, key=lambda item:(item[1], -item[2])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1, 3, 3], [2, 3, 1]]）
96. 已知 x = {1, 2, 3}，那么执行语句 x.add(3) 之后，x的值为\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3}）
97. 已知 x = {1:1}，那么执行语句 x[2] = 2之后，len(x)的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
98. 已知 x = {1:1, 2:2}，那么执行语句 x[2] = 4之后，len(x)的值为\_\_\_\_\_\_\_\_\_\_\_\_。（2）
99. 假设已从标准库functools导入reduce()函数，那么表达式 reduce(lambda x, y: x-y, [1, 2, 3]) 的值为\_\_\_\_。（-4）
100. 假设已从标准库functools导入reduce()函数，那么表达式 reduce(lambda x, y: x+y, [1, 2, 3]) 的值为\_\_\_\_。（6）
101. 假设已从标准库functools导入reduce()函数，那么表达式reduce(lambda x,y:max(x,y), [1,2,3,4,4,5])的值为\_\_\_\_\_\_\_\_\_。（5）
102. 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）
103. 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]）
104. 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]）
105. 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）
106. 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）
107. 已知列表 x = [1, 2]，执行语句 y = x 后，表达式 x is y 的值为\_\_\_\_\_\_\_。（True）
108. 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）
109. 表达式 sorted(random.sample(range(5), 5)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 1, 2, 3, 4]）
110. 表达式 [i for i in range(10) if i>8] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[9]）
111. 已知有列表 x = [[1, 2, 3], [4, 5, 6]]，那么表达式 [[row[i] for row in x] for i in range(len(x[0]))] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1, 4], [2, 5], [3, 6]]）
112. 执行语句 x,y,z = map(str, range(3)) 之后，变量y的值为\_\_\_\_\_\_\_\_\_\_\_。（'1'）
113. 已知列表 x = [1, 2]，那么执行语句 x.extend([3]) 之后， x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3]）
114. 已知列表 x = [1, 2]，那么执行语句 x.append([3]) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, [3]]）
115. 表达式 print(0b10101) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（21）
116. 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]）
117. 已知 x = range(1,4) 和 y = range(4,7)，那么表达式 sum([i\*j for i,j in zip(x,y)]) 的值为\_\_\_\_\_\_\_\_\_\_。（32）
118. 表达式 [5 for i in range(3)] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[5, 5, 5]）
119. 表达式 {1, 2, 3} == {1, 3, 2} 的值为\_\_\_\_\_\_\_\_\_\_\_。（True）
120. 表达式 [1, 2, 3] == [1, 3, 2] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（False）
121. 已知 x = [1, 2, 1]，那么表达式 id(x[0]) == id(x[2]) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
122. 表达式 3 not in [1, 2, 3]的值为\_\_\_\_\_\_\_\_\_\_。（False）
123. 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]）
124. 已知 x = [1, 2]，那么执行语句 x[0:1] = [3, 3]之后，x的值为\_\_\_\_\_\_\_\_\_\_\_。（[3, 3, 2]）
125. 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]）
126. 已知 x = [[1, 2, 3,], [4, 5, 6]]，那么表达式 sum([i\*j for i,j in zip(\*x)]) 的值为\_\_\_\_\_\_。（32）
127. 已知列表 x = [1, 2, 3] 和 y = [4, 5, 6]，那么表达式 [(i,j) for i, j in zip(x,y) if i==3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[(3, 6)]）
128. 已知列表 x = [1.0, 2.0, 3.0]，那么表达式 sum(x)/len(x) 的值为\_\_\_\_\_\_\_\_\_\_\_。（2.0）
129. 已知 x = {1:2, 2:3, 3:4}，那么表达式 sum(x) 的值为\_\_\_\_\_\_\_\_\_\_\_。（6）
130. 已知 x = {1:2, 2:3, 3:4}，那么表达式 sum(x.values()) 的值为\_\_\_\_\_\_\_\_\_。（9）
131. 已知 x = [3, 2, 3, 3, 4]，那么表达式 [index for index, value in enumerate(x) if value==3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 2, 3]）
132. 表达式 1234%1000//100 的值为\_\_\_\_\_\_\_\_\_\_\_。（2）
133. 表达式 3 // 5 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
134. 表达式 [1, 2] + [3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3]）
135. 表达式 (1,) + (2,) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（(1, 2)）
136. 表达式 (1) + (2) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（3）
137. 已知 x, y = map(int, ['1', '2'])，那么表达式 x + y 的值为\_\_\_\_\_\_\_。（3）
138. 已知列表 x = list(range(5))，那么执行语句 x.remove(3) 之后，表达式 x.index(4) 的值为\_\_\_\_\_\_\_\_。（3）
139. 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]）
140. 已知列表 x = [1, 3, 2]，那么执行语句 x = x.reverse() 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（None）
141. 已知x为非空列表，那么表达式 x.reverse() == list(reversed(x)) 的值为\_\_\_\_\_\_\_\_\_\_。（False）
142. 已知x为非空列表，那么表达式 x.sort() == sorted(x) 的值为\_\_\_\_\_\_\_\_\_\_。（False）
143. 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]）
144. 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]）
145. 已知列表x中包含超过5个以上的元素，那么表达式 x == x[:5]+x[5:] 的值为\_\_\_\_\_\_\_\_。（True）
146. 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）
147. 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）
148. 已知字典 x = {i:str(i+3) for i in range(3)}，那么表达式 ''.join([item[1] for item in x.items()]) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（'345'）
149. 已知列表 x = [1, 3, 2]，那么表达式 [value for index, value in enumerate(x) if index==2] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[2]）
150. 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）
151. 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'）
152. 表达式 set([1,2,3]) == {1, 2, 3} 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
153. 表达式 set([1,2, 2,3]) == {1, 2, 3} 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
154. 表达式 '%c'%65 == str(65) 的值为\_\_\_\_\_\_\_\_\_\_\_。（False）
155. 表达式 '%s'%65 == str(65) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
156. 表达式 chr(ord('b')^32) 的值为\_\_\_\_\_\_\_\_\_。（'B'）
157. 表达式 'abc' in 'abdcefg' 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
158. 已知x为整数变量，那么表达式 int(hex(x), 16) == x 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
159. 已知 x, y = 3, 5，那么执行x, y = y, x 之后，x的值为\_\_\_\_\_\_\_\_\_\_。（5）
160. 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]）
161. 已知x = list(range(20))，那么表达式x[-1]的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（19）
162. 已知x = 3+4j和y = 5+6j，那么表达式x+y的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（8+10j）
163. 已知x = [3]，那么执行x += [5]之后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[3, 5]）
164. 已知x = [3, 3, 4]，那么表达式id(x[0])==id(x[1])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
165. 表达式int('11', 2)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）
166. 表达式int('11', 8)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（9）
167. 表达式int(bin(54321), 2)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（54321）
168. 表达式chr(ord('A')+1)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'B'）
169. 表达式int(str(34)) == 34的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
170. 表达式list(str([3, 4])) == [3, 4]的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
171. 表达式{1, 2, 3, 4, 5, 6} ^ {5, 6, 7, 8}的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{1, 2, 3, 4, 7, 8}）
172. 表达式15 // 4的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）
173. 表达式sorted({'a':3, 'b':9, 'c':78})的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['a', 'b', 'c']）
174. 表达式sorted({'a':3, 'b':9, 'c':78}.values())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（[3, 9, 78]）
175. 已知x = [3, 2, 4, 1]，那么执行语句x = x.sort()之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（None）
176. 表达式list(filter(lambda x: x>5, range(10)))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[6, 7, 8, 9]）
177. 已知x = list(range(20))，那么语句print(x[100:200])的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[]）
178. 已知x = list(range(20))，那么执行语句x[:18] = []后列表x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[18, 19]）
179. 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]）
180. 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]）
181. 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]）
182. 已知x = [[]] \* 3，那么执行语句x[0].append(1)之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1], [1], [1]]）
183. 已知x = [[] for i in range(3)]，那么执行语句x[0].append(1)之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[[1], [], []]）
184. 已知x = ([1], [2])，那么执行语句x[0].append(3)后x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（([1, 3], [2])）
185. 已知x = {1:1, 2:2}，那么执行语句x.update({2:3, 3:3})之后，表达式sorted(x.items())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[(1, 1), (2, 3), (3, 3)]）
186. 已知x = {1:1, 2:2}，那么执行语句x[3] = 3之后，表达式sorted(x.items())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[(1, 1), (2, 2), (3, 3)]）
187. 已知x = [1, 2, 3]，那么表达式not (set(x\*100)-set(x))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
188. 已知x = [1, 2, 3]，那么表达式not (set(x\*100)&set(x))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
189. 表达式{'x': 1, \*\*{'y': 2}}的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{'x': 1, 'y': 2}）
190. 表达式{\*range(4), 4, \*(5, 6, 7)}的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（{0, 1, 2, 3, 4, 5, 6, 7}）
191. 已知 x = [1,2,3,4,5]，那么执行语句 x[::2] = range(3) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 2, 1, 4, 2]）
192. 已知 x = [1,2,3,4,5]，那么执行语句 x[::2] = map(lambda y:y!=5,range(3)) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[True, 2, True, 4, True]）
193. 已知 x = [1,2,3,4,5]，那么执行语句 x[1::2] = sorted(x[1::2], reverse=True) 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 4, 3, 2, 5]）
194. 表达式 True\*3 的值为\_\_\_\_\_\_\_\_\_\_。（3）
195. 表达式 False+1 的值为\_\_\_\_\_\_\_\_\_\_。（1）

**第3章 选择结构与循环结构**

1. 表达式 'ab' in 'acbed' 的值为\_\_\_\_\_\_\_\_。（False）
2. 假设n为整数，那么表达式 n&1 == n%2 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
3. 关键字\_\_\_\_\_\_\_\_\_\_用于测试一个对象是否是一个可迭代对象的元素。（in）
4. 表达式 3<5>2 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
5. 已知 x = {'a':'b', 'c':'d'}，那么表达式 'a' in x 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
6. 已知 x = {'a':'b', 'c':'d'}，那么表达式 'b' in x 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
7. 已知 x = {'a':'b', 'c':'d'}，那么表达式 'b' in x.values() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
8. 表达式 1<2<3 的值为\_\_\_\_\_\_\_\_\_。（True）
9. 表达式 3 or 5 的值为\_\_\_\_\_\_\_\_。（3）
10. 表达式 0 or 5 的值为\_\_\_\_\_\_\_\_\_。（5）
11. 表达式 3 and 5 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（5）
12. 表达式 3 and not 5 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
13. Python中用于表示逻辑与、逻辑或、逻辑非运算的关键字分别是\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_。（and、or、not）
14. Python 3.x语句 for i in range(3):print(i, end=',') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0,1,2,）
15. Python 3.x语句 print(1, 2, 3, sep=',') 的输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1,2,3）
16. 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）
17. 在循环语句中，\_\_\_\_\_\_\_\_\_\_语句的作用是提前结束本层循环。（break）
18. 在循环语句中，\_\_\_\_\_\_\_语句的作用是提前进入下一次循环。（continue）
19. 表达式 5 if 5>6 else (6 if 3>2 else 5) 的值为\_\_\_\_\_\_\_\_\_。（6）
20. Python关键字elif表示\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_\_两个单词的缩写。（else、if）
21. 表达式 3 in {1, 2, 3} 的值为\_\_\_\_\_\_\_\_\_。（True）
22. 表达式 'ac' in 'abce' 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
23. 表达式 not 3 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）

**第4章 字符串与正则表达式**

1. 表达式 'abc' in ('abcdefg') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
2. 表达式 'abc' in ['abcdefg'] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
3. 表达式 '\x41' == 'A' 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
4. Python语句''.join(list('hello world!'))执行的结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'hello world!'）
5. 转义字符r’\n’的含义是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（回车换行）
6. 已知列表对象x = ['11', '2', '3']，则表达式 max(x) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'3'）
7. 表达式 min(['11', '2', '3']) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'11'）
8. 已知列表对象x = ['11', '2', '3']，则表达式max(x, key=len) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'11'）
9. 已知 path = r'c:\test.html'，那么表达式 path[:-4]+'htm' 的值为\_\_\_\_\_\_\_\_\_\_。（'c:\\test.htm'）
10. 表达式 list(str([1,2,3])) == [1,2,3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
11. 表达式 str([1, 2, 3]) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'[1, 2, 3]'）
12. 表达式 str((1, 2, 3)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'(1, 2, 3)'）
13. 表达式 sum(range(1, 10, 2)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（25）
14. 表达式 sum(range(1, 10)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（45）
15. 表达式 '%c'%65 的值为\_\_\_\_\_\_\_\_\_。（'A'）
16. 表达式 '%s'%65 的值为\_\_\_\_\_\_\_\_\_\_。（'65'）
17. 表达式 '%d,%c' % (65, 65) 的值为\_\_\_\_\_\_\_\_。（'65,A'）
18. 表达式 'The first:{1}, the second is {0}'.format(65,97) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'The first:97, the second is 65'）
19. 表达式 '{0:#d},{0:#x},{0:#o}'.format(65) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（'65,0x41,0o101'）
20. 表达式 isinstance('abcdefg', str) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
21. 表达式 isinstance('abcdefg', object) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
22. 表达式 isinstance(3, object) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
23. 表达式 'abcabcabc'.rindex('abc') 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（6）
24. 表达式 ':'.join('abcdefg'.split('cd')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'ab:efg'）
25. 表达式 'Hello world. I like Python.'.rfind('python') 的值为\_\_\_\_\_\_\_\_。（-1）
26. 表达式 'abcabcabc'.count('abc') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）
27. 表达式 'apple.peach,banana,pear'.find('p') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1）
28. 表达式 'apple.peach,banana,pear'.find('ppp') 的值为\_\_\_\_\_\_\_\_。（-1）
29. 表达式 'abcdefg'.split('d') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['abc', 'efg']）
30. 表达式 ':'.join('1,2,3,4,5'.split(',')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'1:2:3:4:5'）
31. 表达式 ','.join('a b ccc\n\n\nddd '.split()) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'a,b,ccc,ddd'）
32. 表达式 'Hello world'.upper() 的值为\_\_\_\_\_\_\_\_\_\_\_。（'HELLO WORLD'）
33. 表达式 'Hello world'.lower() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（'hello world'）
34. 表达式 'Hello world'.lower().upper() 的值为\_\_\_\_\_\_\_\_\_\_\_。（'HELLO WORLD'）
35. 表达式 'Hello world'.swapcase().swapcase() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'Hello world'）
36. 表达式 r'c:\windows\notepad.exe'.endswith('.exe') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
37. 表达式 r'c:\windows\notepad.exe'.endswith(('.jpg', '.exe')) 的值为\_\_\_\_\_\_\_。（True）
38. 表达式 'C:\\Windows\\notepad.exe'.startswith('C:') 的值为\_\_\_\_\_\_\_\_\_。（True）
39. 表达式 len('Hello world!'.ljust(20)) 的值为\_\_\_\_\_\_\_\_\_。（20）
40. 表达式 len('abcdefg'.ljust(3)) 的值为\_\_\_\_\_\_\_\_\_。（7）
41. 表达式 'a' + 'b' 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（'ab'）
42. 已知 x = '123' 和 y = '456'，那么表达式 x + y 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'123456'）
43. 表达式 'a'.join('abc'.partition('a')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'aaabc'）
44. 表达式 re.split('\.+', 'alpha.beta...gamma..delta') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['alpha', 'beta', 'gamma', 'delta']）
45. 已知 x = 'a234b123c'，并且re模块已导入，则表达式 re.split('\d+', x) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['a', 'b', 'c']）
46. 表达式 ''.join('asdssfff'.split('sd')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（'assfff'）
47. 表达式 ''.join(re.split('[sd]','asdssfff')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'afff'）
48. 假设re模块已导入，那么表达式 re.findall('(\d)\\1+', '33abcd112') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['3', '1']）
49. 语句 print(re.match('abc', 'defg')) 输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_。（None）
50. 表达式 'Hello world!'[-4] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'r'）
51. 表达式 'Hello world!'[-4:] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'rld!'）
52. 表达式 'test.py'.endswith(('.py', '.pyw')) 的值为\_\_\_\_\_\_\_\_\_\_。（True）
53. 表达式 len('abc'.ljust(20)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（20）
54. 代码 print(re.match('^[a-zA-Z]+$','abcDEFG000')) 的输出结果为\_\_\_\_\_\_\_\_\_。（None）
55. 当在字符串前加上小写字母\_\_\_\_\_或大写字母\_\_\_\_\_表示原始字符串，不对其中的任何字符进行转义。（r、R）
56. 在设计正则表达式时，字符\_\_\_\_\_\_\_紧随任何其他限定符(\*、+、?、{n}、{n,}、{n,m})之后时，匹配模式是“非贪心的”，匹配搜索到的、尽可能短的字符串。（?）
57. 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'）
58. 表达式 len('中国'.encode('utf-8')) 的值为\_\_\_\_\_\_\_\_\_\_\_。（6）
59. 表达式 len('中国'.encode('gbk')) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（4）
60. 表达式 chr(ord('A')+2) 的值为\_\_\_\_\_\_\_\_\_\_。（'C'）
61. 表达式 'abcab'.replace('a','yy') 的值为\_\_\_\_\_\_\_\_\_\_\_。（'yybcyyb'）
62. 已知 table = ''.maketrans('abcw', 'xyzc')，那么表达式 'Hellow world'.translate(table) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'Helloc corld'）
63. 表达式 'hello world, hellow every one'.replace('hello', 'hi') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'hi world, hiw every one'）
64. 已知字符串 x = 'hello world'，那么执行语句 x.replace('hello', 'hi') 之后，x的值为\_\_\_\_\_\_\_\_\_\_\_\_。（'hello world'）
65. 正则表达式元字符\_\_\_\_\_\_\_\_用来表示该符号前面的字符或子模式1次或多次出现。（+）
66. 已知 x = 'a b c d'，那么表达式 ','.join(x.split()) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'a,b,c,d'）
67. 正则表达式元字符\_\_\_\_\_\_\_\_用来表示该符号前面的字符或子模式0次或多次出现。（\*）
68. 表达式 'abcab'.strip('ab') 的值为\_\_\_\_\_\_\_\_\_\_。（'c'）
69. 表达式 [str(i) for i in range(3)] 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（['0', '1', '2']）
70. 表达式 'abc.txt'.endswith(('.txt', '.doc', '.jpg')) 的值为\_\_\_\_\_\_\_\_\_\_\_。（True）
71. 代码 print(1,2,3,sep=':') 的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1:2:3）
72. 代码 for i in range(3):print(i, end=',') 的执行结果为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0,1,2,）
73. 表达式 eval('''\_\_import\_\_('math').sqrt(9)''') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（3.0）
74. 表达式 eval('''\_\_import\_\_('math').sqrt(3\*\*2+4\*\*2)''') 的值为\_\_\_\_\_\_\_\_\_。（5.0）
75. 表达式 eval('3+5') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（8）
76. 表达式 eval('[1, 2, 3]') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3]）
77. 假设math标准库已导入，那么表达式 eval('math.sqrt(4)') 的值为\_\_\_\_\_\_\_\_\_。（2.0）
78. 已知x为非空列表，那么表达式 random.choice(x) in x 的值为\_\_\_\_\_\_\_\_\_\_\_。（True）
79. 表达式 'abc10'.isalnum() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
80. 表达式 'abc10'.isalpha() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
81. 表达式 'abc10'.isdigit() 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
82. 表达式 'C:\\windows\\notepad.exe'.endswith('.exe') 的值为\_\_\_\_\_\_\_\_。（True）
83. 表达式 '%s'%[1,2,3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（'[1, 2, 3]'）
84. 表达式 'aaasdf'.lstrip('as') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'df'）
85. 表达式 'aaasdf'.lstrip('af') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'sdf'）
86. 表达式 'aaasdf'.strip('af') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'sd'）
87. 表达式 'aaasdf'.rstrip('af') 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'aaasd'）
88. 表达式 len('SDIBT') 的值为\_\_\_\_\_\_\_\_\_\_。（5）
89. 表达式 'Hello world!'.count('l') 的值为\_\_\_\_\_\_\_\_\_\_\_。（3）
90. 已知 x = 'abcdefg'，则表达式 x[3:] + x[:3] 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'defgabc'）
91. 字符串编码格式UTF8使用\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_个字节表示一个汉字。（3）
92. 字符串编码格式GBK使用\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_个字节表示一个汉字。（2）
93. 已知字符串编码格式utf8使用3个字节表示一个汉字、1个字节表示英语字母，那么表达式len('abc你好')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（5）
94. 已知字符串编码格式utf8使用3个字节表示一个汉字、1个字节表示英语字母，那么表达式len('abc你好'.encode())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（9）
95. 已知字符串编码格式gbk使用2个字节表示一个汉字、1个字节表示英语字母，那么表达式len('abc你好'.encode('gbk'))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（7）
96. 已知ord('A')的值为65并且hex(65)的值为'0x41'，那么表达式'\x41b'的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'Ab'）
97. 已知formatter = 'good {0}'.format，那么表达式list(map(formatter, ['morning']))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（['good morning']）
98. 已知x = 'hello world.'，那么表达式x.find('x')和x.rfind('x')的值都为\_\_\_\_\_\_\_\_\_\_\_\_\_。（-1）
99. 表达式':'.join('hello world.'.split())的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'hello:world.'）
100. 表达式':'.join('a b c d'.split(maxsplit=2))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'a:b:c d'）
101. 已知x = 'hello world'，那么表达式x.replace('l', 'g')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'heggo worgd'）
102. 假设已成功导入Python标准库string，那么表达式len(string.digits)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（10）
103. 表达式'aaaassddf'.strip('af')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'ssdd'）
104. 表达式len('aaaassddf'.strip('afds'))的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
105. 表达式len('hello world'[100:])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
106. 表达式chr(ord('a')^32^32)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'a'）
107. 表达式chr(ord('a')^32)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'A'）
108. 已知x = 'aa b ccc dddd'，那么表达式''.join([v for i,v in enumerate(x[:-1]) if v==x[i+1]])的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'accddd'）
109. 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'）
110. 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']）
111. 假设正则表达式模块re已正确导入，那么表达式re.sub('(.\s)\\1+', '\\1','a a a a a bb')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（'a bb'）
112. 表达式 eval('\*'.join(map(str, range(1, 6)))) 的值为\_\_\_\_\_\_\_\_\_。（120）
113. 正则表达式模块re的\_\_\_\_\_\_\_\_\_\_方法用来编译正则表达式对象。（compile()）
114. 正则表达式模块re的\_\_\_\_\_\_\_\_\_\_\_\_\_\_方法用来在字符串开始处进行指定模式的匹配。（match()）
115. 正则表达式模块re的\_\_\_\_\_\_\_\_\_\_\_\_\_\_方法用来在整个字符串中进行指定模式的匹配。（search()）
116. 表达式 re.search(r'\w\*?(?P<f>\b\w+\b)\s+(?P=f)\w\*?', 'Beautiful is is better than ugly.').group(0) 的值为\_\_\_\_\_\_\_\_\_\_\_。（'is is'）
117. 表达式 'Beautiful is better than ugly.'.startswith('Be', 5) 的值为\_\_\_\_\_\_\_\_\_。（False）
118. 已知字典 x = {i:str(i+3) for i in range(3)}，那么表达式 ''.join(x.values()) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（'345'）

**第5章 函数定义与使用**

1. Python中定义函数的关键字是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（def）
2. 在函数内部可以通过关键字\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_来定义全局变量。（global）
3. 如果函数中没有return语句或者return语句不带任何返回值，那么该函数的返回值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（None）
4. 表达式 sum(range(10)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（45）
5. 表达式 sum(range(1, 10, 2)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（25）
6. 表达式 list(filter(None, [0,1,2,3,0,0])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3]）
7. 表达式 list(filter(lambda x:x>2, [0,1,2,3,0,0])) 的值为\_\_\_\_\_\_\_\_\_。（[3]）
8. 表达式 list(range(50, 60, 3)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[50, 53, 56, 59]）
9. 表达式 list(filter(lambda x: x%2==0, range(10))) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（[0, 2, 4, 6, 8]）
10. 表达式 list(filter(lambda x: len(x)>3, ['a', 'b', 'abcd'])) 的值为\_\_\_\_\_\_\_\_\_\_\_。（['abcd']）
11. 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）
12. 表达式 list(map(lambda x: len(x), ['a', 'bb', 'ccc'])) 的值为\_\_\_\_\_\_\_\_\_\_。（[1, 2, 3]）
13. 已知 f = lambda x: x+5，那么表达式 f(3) 的值为\_\_\_\_\_\_\_\_。（8）
14. 表达式 sorted(['abc', 'acd', 'ade'], key=lambda x:(x[0],x[2])) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（['abc', 'acd', 'ade']）
15. 已知函数定义def demo(x, y, op):return eval(str(x)+op+str(y))，那么表达式demo(3, 5, '+')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（8）
16. 已知函数定义def demo(x, y, op):return eval(str(x)+op+str(y))，那么表达式demo(3, 5, '\*')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（15）
17. 已知函数定义def demo(x, y, op):return eval(str(x)+op+str(y))，那么表达式demo(3, 5, '-')的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（-2）
18. 已知f = lambda n: len(bin(n)[bin(n).rfind('1')+1:])，那么表达式f(6)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1）
19. 已知f = lambda n: len(bin(n)[bin(n).rfind('1')+1:])，那么表达式f(7)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（0）
20. 已知 g = lambda x, y=3, z=5: x+y+z，那么表达式 g(2) 的值为\_\_\_\_\_\_\_\_。（10）
21. 已知函数定义 def func(\*p):return sum(p)，那么表达式 func(1,2,3) 的值为\_\_\_\_\_\_。（6）
22. 已知函数定义 def func(\*p):return sum(p)，那么表达式 func(1,2,3, 4) 的值为\_\_\_\_\_\_。（10）
23. 已知函数定义 def func(\*\*p):return sum(p.values())，那么表达式 func(x=1, y=2, z=3) 的值为\_\_\_\_\_\_\_\_。（6）
24. 已知函数定义 def func(\*\*p):return ''.join(sorted(p))，那么表达式 func(x=1, y=2, z=3)的值为\_\_\_\_\_\_\_\_\_\_。（'xyz'）
25. 已知 f = lambda x: 5，那么表达式 f(3)的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（5）

**第6章 面向对象程序设计**

1. Python使用\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_关键字来定义类。（class）
2. 表达式 isinstance('abc', str) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
3. 表达式 isinstance('abc', int) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
4. 表达式 isinstance(4j, (int, float, complex)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
5. 表达式 isinstance('4', (int, float, complex)) 的值为\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
6. 表达式 type(3) in (int, float, complex) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
7. 表达式 type(3.0) in (int, float, complex) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
8. 表达式 type(3+4j) in (int, float, complex) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（True）
9. 表达式 type('3') in (int, float, complex) 的值为\_\_\_\_\_\_\_\_\_\_\_\_。（False）
10. 表达式 type(3) == int 的值为\_\_\_\_\_\_\_\_\_\_。（True）
11. 在Python定义类时，与运算符“\*\*”对应的特殊方法名为 \_\_\_\_\_\_\_\_\_\_\_\_\_。（\_\_pow\_\_()）
12. 在Python中定义类时，与运算符“//”对应的特殊方法名为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（\_\_floordiv\_\_()）
13. 表达式type({}) == dict的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（True）
14. 表达式type({}) == set的值为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（False）
15. 在Python中，不论类的名字是什么，构造方法的名字都是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（\_\_init\_\_）
16. 如果在设计一个类时实现了\_\_contains\_\_ ()方法，那么该类的对象会自动支持\_\_\_\_\_\_\_\_\_\_\_\_\_运算符。（in）

**第7章 文件操作**

1. 对文件进行写入操作之后，\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_方法用来在不关闭文件对象的情况下将缓冲区内容写入文件。（flush()）
2. Python内置函数\_\_\_\_\_\_\_\_\_\_\_\_\_用来打开或创建文件并返回文件对象。（open()）
3. 使用上下文管理关键字\_\_\_\_\_\_\_\_\_\_\_\_\_\_可以自动管理文件对象，不论何种原因结束该关键字中的语句块，都能保证文件被正确关闭。（with）
4. Python标准库os中用来列出指定文件夹中的文件和子文件夹列表的方式是\_\_\_\_\_\_\_\_\_\_\_\_。（listdir()）
5. Python标准库os.path中用来判断指定文件是否存在的方法是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（exists()）
6. Python标准库os.path中用来判断指定路径是否为文件的方法是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（isfile()）
7. Python标准库os.path中用来判断指定路径是否为文件夹的方法是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（isdir()）
8. Python标准库os.path中用来分割指定路径中的文件扩展名的方法是\_\_\_\_\_\_\_\_\_\_。（splitext()）
9. Python扩展库\_\_\_\_\_\_\_\_\_\_\_\_\_支持Excel 2007或更高版本文件的读写操作。（openpyxl）
10. Python标准库\_\_\_\_\_\_\_\_\_\_\_\_中提供了计算MD5摘要的方法md5()。（hashlib）
11. 已知当前文件夹中有纯英文文本文件readme.txt，请填空完成功能把readme.txt文件中的所有内容复制到dst.txt中，with open('readme.txt') as src, open('dst.txt', \_\_\_\_\_\_\_\_\_\_\_\_) as dst:dst.write(src.read())。（'w'）

**第8章 异常处理结构**

1. Python内建异常类的基类是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（BaseException）

**第9章 GUI编程**

**第10章 网络编程**

1. Python标准库\_\_\_\_\_\_\_\_\_\_对Socket进行了二次封装，支持Socket接口的访问，大幅度简化了网络程序的开发。（socket）

**第13章 多线程编程**

1. Python扩展库\_\_\_\_\_\_\_\_\_\_\_\_\_\_中封装了Windows底层几乎所有API函数。（pywin32）
2. 线程对象的\_\_\_\_\_\_\_\_\_\_\_方法用来阻塞当前线程，指定线程运行结束或超时后继续运行当前线程。（join()）

**第14章 数据库编程**

1. Python用来访问和操作内置数据库SQLite的标准库是\_\_\_\_\_\_\_\_\_\_\_\_\_。（sqlite3）
2. 用于删除数据库表test中所有name字段值为’10001’的记录的SQL语句为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（delete from test where name=’10001’）

**第15章 多媒体编程**

1. Python扩展库\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_完美封装了图形库OpenGL的功能。（pyopengl）
2. Python扩展库\_\_\_\_\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_\_\_\_\_提供了图像处理功能。（PIL、pillow）
3. 判断题

**第1章 基础知识**

1. Python是一种跨平台、开源、免费的高级动态编程语言。（对）
2. Python 3.x完全兼容Python 2.x。（错）
3. Python 3.x和Python 2.x唯一的区别就是：print在Python 2.x中是输出语句，而在Python 3.x中是输出函数。（错）
4. 在Windows平台上编写的Python程序无法在Unix平台运行。（错）
5. 不可以在同一台计算机上安装多个Python版本。（错）
6. 已知 x = 3，那么赋值语句 x = 'abcedfg' 是无法正常执行的。（错）
7. Python变量使用前必须先声明，并且一旦声明就不能再当前作用域内改变其类型。（错）
8. Python采用的是基于值得自动内存管理方式。（对）
9. 在任何时刻相同的值在内存中都只保留一份（错）
10. Python不允许使用关键字作为变量名，允许使用内置函数名作为变量名，但这会改变函数名的含义。（对）
11. 在Python中可以使用if作为变量名。（错）
12. 在Python 3.x中可以使用中文作为变量名。（对）
13. Python变量名必须以字母或下划线开头，并且区分字母大小写。（对）
14. 加法运算符可以用来连接字符串并生成新字符串。（对）
15. 9999\*\*9999这样的命令在Python中无法运行。（错）
16. 3+4j不是合法的Python表达式。（错）
17. 0o12f是合法的八进制数字。（错）
18. 不管输入什么，Python 3.x中input()函数的返回值总是字符串。（对）
19. pip命令也支持扩展名为.whl的文件直接安装Python扩展库。（对）
20. 只有Python扩展库才需要导入以后才能使用其中的对象，Python标准库不需要导入即可使用其中的所有对象和方法。（错）
21. 在Python中0xad是合法的十六进制数字表示形式。（对）
22. 3+4j 是合法Python数字类型。（对）
23. 在Python中0oa1是合法的八进制数字表示形式。（错）
24. Python使用缩进来体现代码之间的逻辑关系。（对）
25. Python代码的注释只有一种方式，那就是使用#符号。（错）
26. 放在一对三引号之间的任何内容将被认为是注释。（错）
27. 尽管可以使用import语句一次导入任意多个标准库或扩展库，但是仍建议每次只导入一个标准库或扩展库。（对）
28. 为了让代码更加紧凑，编写Python程序时应尽量避免加入空格和空行。（错）
29. 在Python 3.5中运算符+不仅可以实现数值的相加、字符串连接，还可以实现列表、元组的合并和集合的并集运算。（错）
30. 在Python中可以使用 for 作为变量名。（错）
31. 在Python中可以使用 id 作为变量名，尽管不建议这样做。（对）
32. Python关键字不可以作为变量名。（对）
33. 一个数字5也是合法的Python表达式。（对）
34. 执行语句 from math import sin 之后，可以直接使用 sin() 函数，例如 sin(3)。（对）
35. 不同版本的Python不能安装到同一台计算机上。（错）
36. 一般来说，Python扩展库没有通用于所有版本Python的，安装时应选择与已安装Python的版本对应的扩展库。（对）
37. Python变量名区分大小写，所以student和Student不是同一个变量。（对）
38. 在Python 3.x中reduce()是内置函数。（错）
39. 如果只需要math模块中的sin()函数，建议使用from math import sin来导入，而不要使用import math导入整个模块。（对）
40. 在Python 3.x中，使用内置函数input()接收用户输入时，不论用户输入的什么格式，一律按字符串进行返回。（对）
41. 安装Python扩展库时只能使用pip工具在线安装，如果安装不成功就没有别的办法了。（错）

**第2章 Python序列**

1. Python支持使用字典的“键”作为下标来访问字典中的值。（对）
2. 列表可以作为字典的“键”。（错）
3. 元组可以作为字典的“键”。（对）
4. 字典的“键”必须是不可变的。（对）
5. 已知x为非空列表，那么表达式 sorted(x, reverse=True) == list(reversed(x)) 的值一定是True。（错）
6. 已知x为非空列表，那么x.sort(reverse=True)和x.reverse()的作用是等价的。（错）
7. 生成器推导式比列表推导式具有更高的效率，推荐使用。（对）
8. Python集合中的元素不允许重复。（对）
9. Python集合可以包含相同的元素。（错）
10. Python字典中的“键”不允许重复。（对）
11. Python字典中的“值”不允许重复。（错）
12. Python集合中的元素可以是元组。（对）
13. Python集合中的元素可以是列表。（错）
14. Python字典中的“键”可以是列表。（错）
15. Python字典中的“键”可以是元组。（对）
16. Python列表中所有元素必须为相同类型的数据。（错）
17. Python列表、元组、字符串都属于有序序列。（对）
18. 已知A和B是两个集合，并且表达式A<B的值为False，那么表达式A>B的值一定为True。（错）
19. 列表对象的append()方法属于原地操作，用于在列表尾部追加一个元素。（对）
20. 对于列表而言，在尾部追加元素比在中间位置插入元素速度更快一些，尤其是对于包含大量元素的列表。（对）
21. 假设有非空列表x，那么x.append(3)、x = x+[3]与x.insert(0,3)在执行时间上基本没有太大区别。（错）
22. 使用Python列表的方法insert()为列表插入元素时会改变列表中插入位置之后元素的索引。（对）
23. 假设x为列表对象，那么x.pop()和x.pop(-1)的作用是一样的。（对）
24. 使用del命令或者列表对象的remove()方法删除列表中元素时会影响列表中部分元素的索引。（对）
25. 已知列表 x = [1, 2, 3]，那么执行语句 x = 3 之后，变量x的地址不变。（错）
26. 使用列表对象的remove()方法可以删除列表中首次出现的指定元素，如果列中不存在要删除的指定元素则抛出异常。（对）
27. 元组是不可变的，不支持列表对象的inset()、remove()等方法，也不支持del命令删除其中的元素，但可以使用del命令删除整个元组对象。（对）
28. Python字典和集合属于无序序列。（对）
29. 无法删除集合中指定位置的元素，只能删除特定值的元素。（对）
30. 元组的访问速度比列表要快一些，如果定义了一系列常量值，并且主要用途仅仅是对其进行遍历二不需要进行任何修改，建议使用元组而不使用列表。（对）
31. 当以指定“键”为下标给字典对象赋值时，若该“键”存在则表示修改该“键”对应的“值”，若不存在则表示为字典对象添加一个新的“键-值对”。（对）
32. 假设x是含有5个元素的列表，那么切片操作x[10:]是无法执行的，会抛出异常。（错）
33. 只能对列表进行切片操作，不能对元组和字符串进行切片操作。（错）
34. 只能通过切片访问列表中的元素，不能使用切片修改列表中的元素。（错）
35. 只能通过切片访问元组中的元素，不能使用切片修改元组中的元素。（对）
36. 字符串属于Python有序序列，和列表、元组一样都支持双向索引。（对）
37. Python字典和集合支持双向索引。（错）
38. Python集合不支持使用下标访问其中的元素。（对）
39. 相同内容的字符串使用不同的编码格式进行编码得到的结果并不完全相同。（对）
40. 删除列表中重复元素最简单的方法是将其转换为集合后再重新转换为列表。（对）
41. 已知列表x中包含超过5个以上的元素，那么语句 x = x[:5]+x[5:] 的作用是将列表x中的元素循环左移5位。（错）
42. 对于生成器对象x = (3 for i in range(5))，连续两次执行list(x)的结果是一样的。（错）
43. 对于大量列表的连接，extend()方法比运算符+具有更高的效率。（对）
44. 表达式 {1, 3, 2} > {1, 2, 3} 的值为True。（错）
45. 列表对象的extend()方法属于原地操作，调用前后列表对象的地址不变。（对）
46. 对于数字n，如果表达式 0 not in [n%d for d in range(2, n)] 的值为True则说明n是素数。（对）
47. 表达式 ‘a’+1的值为’b’。（错）
48. 创建只包含一个元素的元组时，必须在元素后面加一个逗号，例如(3,)。（对）
49. 表达式 list('[1, 2, 3]') 的值是[1, 2, 3]。（错）
50. 同一个列表对象中的元素类型可以各不相同。（对）
51. 同一个列表对象中所有元素必须为相同类型。（错）
52. 已知x为非空列表，那么执行语句x[0] = 3之后，列表对象x的内存地址不变。（对）
53. 列表可以作为集合的元素。（错）
54. 集合可以作为列表的元素。（对）
55. 元组可以作为集合的元素。（对）
56. 集合可以作为元组的元素。（对）
57. 字典可以作为集合的元素。（错）
58. 集合可以作为字典的键。（错）
59. 集合可以作为字典的值。（对）
60. 可以使用del删除集合中的部分元素。（错）
61. 列表对象的pop()方法默认删除并返回最后一个元素，如果列表已空则抛出异常。（对）
62. 表达式 {1, 2} \* 2 的值为 {1, 2, 1, 2}。（错）
63. Python字典支持双向索引。（错）
64. Python集合支持双向索引。（错）
65. Python元组支持双向索引。（对）
66. 假设re模块已成功导入，并且有 pattern = re.compile('^'+'\.'.join([r'\d{1,3}' for i in range(4)])+'$')，那么表达式 pattern.match('192.168.1.103') 的值为None。（错）
67. 假设random模块已导入，那么表达式 random.sample(range(10), 20) 的作用是生成20个不重复的整数。（错）
68. 假设random模块已导入，那么表达式 random.sample(range(10), 7) 的作用是生成7个不重复的整数。（对）
69. 使用random模块的函数randint(1, 100)获取随机数时，有可能会得到100。（对）
70. 已知x = (1, 2, 3, 4)，那么执行x[0] = 5之后，x的值为(5, 2, 3, 4)。（错）
71. 已知x = 3，那么执行x += 6语句前后x的内存地址是不变的。（错）
72. 成员测试运算符in作用于集合时比作用于列表快得多。（对）
73. 内置函数len()返回指定序列的元素个数，适用于列表、元组、字符串、字典、集合以及range、zip等迭代对象。（对）
74. 已知x和y是两个等长的整数列表，那么表达式sum((i\*j for i, j in zip(x, y)))的作用是计算这两个列表所表示的向量的内积。（对）
75. 已知x和y是两个等长的整数列表，那么表达式[i+j for i,j in zip(x,y)]的作用时计算这两个列表所表示的向量的和。（对）
76. 表达式int('1'\*64, 2)与sum(2\*\*i for i in range(64))的计算结果是一样的，但是前者更快一些。（对）
77. 已知x = list(range(20))，那么语句del x[::2]可以正常执行。（对）
78. 已知x = list(range(20))，那么语句x[::2] = []可以正常执行。（错）
79. 已知x = list(range(20))，那么语句print(x[100:200])无法正常执行。（错）
80. 已知x是个列表对象，那么执行语句y = x之后，对y所做的任何操作都会同样作用到x上。（对）
81. 已知x是个列表对象，那么执行语句y = x[:]之后，对y所做的任何操作都会同样作用到x上。（错）
82. 在Python中，变量不直接存储值，而是存储值的引用，也就是值在内存中的地址。（对）
83. 表达式(i\*\*2 for i in range(100))的结果是个元组。（错）
84. 在Python中元组的值是不可变的，因此，已知x = ([1], [2])，那么语句x[0].append(3)是无法正常执行的。（错）
85. Python内置的字典dict中元素是按添加的顺序依次进行存储的。（错）
86. Python内置的集合set中元素顺序是按元素的哈希值进行存储的，并不是按先后顺序。（对）
87. 已知x = {1:1, 2:2}，那么语句x[3] =3无法正常执行。（错）
88. Python内置字典是无序的，如果需要一个可以记住元素插入顺序的字典，可以使用collections.OrderedDict。（对）
89. 已知列表x = [1, 2, 3, 4]，那么表达式x.find(5)的值应为-1。（错）
90. 列表对象的排序方法sort()只能按元素从小到大排列，不支持别的排序方式。（错）
91. 已知x是一个列表，那么x = x[3:] + x[:3]可以实现把列表x中的所有元素循环左移3位。（对）

**第3章 选择结构与循环结构**

1. 如果仅仅是用于控制循环次数，那么使用for i in range(20)和for i in range(20, 40)的作用是等价的。（对）
2. 在循环中continue语句的作用是跳出当前循环。（错）
3. 在编写多层循环时，为了提高运行效率，应尽量减少内循环中不必要的计算。（对）
4. 带有else子句的循环如果因为执行了break语句而退出的话，则会执行else子句中的代码。（错）
5. 对于带有else子句的循环语句，如果是因为循环条件表达式不成立而自然结束循环，则执行else子句中的代码。（对）
6. 在条件表达式中不允许使用赋值运算符“=”，会提示语法错误。（对）

**第4章 字符串与正则表达式**

1. 在UTF-8编码中一个汉字需要占用3个字节。（对）
2. 在GBK和CP936编码中一个汉字需要2个字节。（对）
3. 在Python中，任意长的字符串都遵守驻留机制。（错）
4. Python运算符%不仅可以用来求余数，还可以用来格式化字符串。（对）
5. Python字符串方法replace()对字符串进行原地修改。（错）
6. 如果需要连接大量字符串成为一个字符串，那么使用字符串对象的join()方法比运算符+具有更高的效率。（对）
7. 正则表达式模块re的match()方法是从字符串的开始匹配特定模式，而search()方法是在整个字符串中寻找模式，这两个方法如果匹配成功则返回match对象，匹配失败则返回空值None。（对）
8. 已知x为非空字符串，那么表达式 ''.join(x.split()) == x 的值一定为True。（错）
9. 已知x为非空字符串，那么表达式 ','.join(x.split(',')) == x 的值一定为True。（对）
10. 当作为条件表达式时，[]与None等价。（对）
11. 表达式 [] == None 的值为True。（错）
12. 当作为条件表达式时，{}与None等价。（对）
13. 表达式 {}==None 的值为True。（错）
14. 表达式 pow(3,2) == 3\*\*2 的值为True。（对）
15. 当作为条件表达式时，空值、空字符串、空列表、空元组、空字典、空集合、空迭代对象以及任意形式的数字0都等价于False。（对）
16. 正则表达式对象的match()方法可以在字符串的指定位置开始进行指定模式的匹配。（对）
17. 使用正则表达式对字符串进行分割时，可以指定多个分隔符，而字符串对象的split()方法无法做到这一点。（对）
18. 正则表达式元字符“^”一般用来表示从字符串开始处进行匹配，用在一对方括号中的时候则表示反向匹配，不匹配方括号中的字符。（对）
19. 正则表达式元字符“\s”用来匹配任意空白字符。（对）
20. 正则表达式 元字符“\d”用来匹配任意数字字符。（对）
21. 已知x和y是两个字符串，那么表达式sum((1 for i,j in zip(x,y) if i==j))可以用来计算两个字符串中对应位置字符相等的个数。（对）
22. Python 3.x中字符串对象的encode()方法默认使用utf8作为编码方式。（对）
23. 已知x = 'hellow world.'.encode()，那么表达式x.decode('gbk')的值为'hellow world.'。（对）
24. 已知x = 'Python是一种非常好的编程语言'.encode()，那么表达式x.decode('gbk')的值为'Python是一种非常好的编程语言'。（错）
25. 正则表达式'^http'只能匹配所有以'http'开头的字符串。（对）
26. 正则表达式'^\d{18}|\d{15}$'只能检查给定字符串是否为18位或15位数字字符，并不能保证一定是合法的身份证号。（对）
27. 正则表达式'[^abc]'可以一个匹配任意除'a'、'b'、'c'之外的字符。（对）
28. 正则表达式'python|perl'或'p(ython|erl)'都可以匹配'python'或'perl'。（对）
29. 字节串b'hello world'和b'hello world.'的MD5值相差很小。（错）

**第5章 函数设计与应用**

1. 调用函数时，在实参前面加一个型号\*表示序列解包。（对）
2. 在Python 3.x中语句 print(\*[1,2,3]) 不能正确执行。（错）
3. 函数是代码复用的一种方式。（对）
4. 定义函数时，即使该函数不需要接收任何参数，也必须保留一对空的圆括号来表示这是一个函数。（对）
5. 编写函数时，一般建议先对参数进行合法性检查，然后再编写正常的功能代码。（对）
6. 一个函数如果带有默认值参数，那么必须所有参数都设置默认值。（错）
7. 定义Python函数时必须指定函数返回值类型。（错）
8. 定义Python函数时，如果函数中没有return语句，则默认返回空值None。（对）
9. 如果在函数中有语句 return 3，那么该函数一定会返回整数3。（错）
10. 函数中必须包含return语句。（错）
11. 函数中的return语句一定能够得到执行。（错）
12. 不同作用域中的同名变量之间互相不影响，也就是说，在不同的作用域内可以定义同名的变量。（对）
13. 全局变量会增加不同函数之间的隐式耦合度，从而降低代码可读性，因此应尽量避免过多使用全局变量。（对）
14. 函数内部定义的局部变量当函数调用结束后被自动删除。（对）
15. 在函数内部，既可以使用global来声明使用外部全局变量，也可以使用global直接定义全局变量。（对）
16. 在函数内部没有办法定义全局变量。（错）
17. 在函数内部直接修改形参的值并不影响外部实参的值。（对）
18. 在函数内部没有任何方法可以影响实参的值。（错）
19. 调用带有默认值参数的函数时，不能为默认值参数传递任何值，必须使用函数定义时设置的默认值。（错）
20. 在同一个作用域内，局部变量会隐藏同名的全局变量。（对）
21. 形参可以看做是函数内部的局部变量，函数运行结束之后形参就不可访问了。（对）
22. 假设已导入random标准库，那么表达式 max([random.randint(1, 10) for i in range(10)]) 的值一定是10。（错）
23. Python标准库random的方法randint(m,n)用来生成一个[m,n]区间上的随机整数。（对）
24. 在函数内部没有任何声明的情况下直接为某个变量赋值，这个变量一定是函数内部的局部变量。（对）
25. 在Python中定义函数时不需要声明函数参数的类型。（对）
26. 在Python中定义函数时不需要声明函数的返回值类型。（对）
27. 在函数中没有任何办法可以通过形参来影响实参的值。（错）
28. 已知 x = 3，那么执行语句 x+=6 之后，x的内存地址不变。（错）
29. 在定义函数时，某个参数名字前面带有一个\*符号表示可变长度参数，可以接收任意多个普通实参并存放于一个元组之中。（对）
30. 在定义函数时，某个参数名字前面带有两个\*符号表示可变长度参数，可以接收任意多个关键参数并将其存放于一个字典之中。（对）
31. 定义函数时，带有默认值的参数必须出现在参数列表的最右端，任何一个带有默认值的参数右边不允许出现没有默认值的参数。（对）
32. 在调用函数时，可以通过关键参数的形式进行传值，从而避免必须记住函数形参顺序的麻烦。（对）
33. 在调用函数时，必须牢记函数形参顺序才能正确传值。（错）
34. 调用函数时传递的实参个数必须与函数形参个数相等才行。（错）
35. 在编写函数时，建议首先对形参进行类型检查和数值范围检查之后再编写功能代码，或者使用异常处理结构，尽量避免代码抛出异常而导致程序崩溃。（对）
36. lambda表达式中可以使用任意复杂的表达式，但是必须只编写一个表达式。（对）
37. g = lambda x: 3不是一个合法的赋值表达式。（错）
38. 包含yield语句的函数一般成为生成器函数，可以用来创建生成器对象。（对）
39. 在函数中yield语句的作用和return完全一样。（错）
40. 语句pass仅起到占位符的作用，并不会做任何操作。（对）
41. 无法使用lambda表达式定义有名字的函数。（错）

**第6章 面向对象程序设计**

1. 在Python中定义类时，如果某个成员名称前有2个下划线则表示是私有成员。（对）
2. 在类定义的外部没有任何办法可以访问对象的私有成员。（错）
3. Python中一切内容都可以称为对象。（对）
4. 栈和队列的都具有先入后出的特点。（错）
5. 在一个软件的设计与开发中，所有类名、函数名、变量名都应该遵循统一的风格和规范。（对）
6. 定义类时所有实例方法的第一个参数用来表示对象本身，在类的外部通过对象名来调用实例方法时不需要为该参数传值。（对）
7. 在面向对象程序设计中，函数和方法是完全一样的，都必须为所有参数进行传值。（错）
8. Python中没有严格意义上的私有成员。（对）
9. 在Python中定义类时，运算符重载是通过重写特殊方法实现的。例如，在类中实现了\_\_mul\_\_()方法即可支持该类对象的\*\*运算符。（错）
10. 在IDLE交互模式下，一个下划线“\_”表示解释器中最后一次显示的内容或最后一次语句正确执行的输出结果。（对）
11. 对于Python类中的私有成员，可以通过“对象名.\_类名\_\_私有成员名”的方式来访问。（对）
12. 运算符“-”可以用于集合的差集运算。（对）
13. 如果定义类时没有编写析构函数，Python将提供一个默认的析构函数进行必要的资源清理工作。（对）
14. 已知seq为长度大于10的列表，并且已导入random模块，那么[random.choice(seq) for i in range(10)]和random.sample(seq,10)等价。（错）
15. 在派生类中可以通过“基类名.方法名()”的方式来调用基类中的方法。（对）
16. Python支持多继承，如果父类中有相同的方法名，而在子类中调用时没有指定父类名，则Python解释器将从左向右按顺序进行搜索。（对）
17. 在Python中定义类时实例方法的第一个参数名称必须是self。（错）
18. 在Python中定义类时实例方法的第一个参数名称不管是什么，都表示对象自身。（对）
19. 定义类时如果实现了\_\_contains\_\_()方法，该类对象即可支持成员测试运算in。（对）
20. 定义类时如果实现了\_\_len\_\_()方法，该类对象即可支持内置函数len()。（对）
21. 定义类时实现了\_\_eq\_\_()方法，该类对象即可支持运算符==。（对）
22. 定义类时实现了\_\_pow\_\_()方法，该类对象即可支持运算符\*\*。（对）
23. Python类的构造函数是\_\_init\_\_()。（对）
24. 定义类时，在一个方法前面使用@classmethod进行修饰，则该方法属于类方法。（对）
25. 定义类时，在一个方法前面使用@staticmethod进行休息，则该方法属于静态方法。（对）
26. 通过对象不能调用类方法和静态方法。（错）
27. 在Python中可以为自定义类的对象动态增加新成员。（对）
28. Python类不支持多继承。（错）
29. 属性可以像数据成员一样进行访问，但赋值时具有方法的优点，可以对新值进行检查。（对）
30. 只可以动态为对象增加数据成员，而不能为对象动态增加成员方法。（错）
31. 任何包含\_\_call\_\_()方法的类的对象都是可调用的。（对）
32. 在Python中函数和类都属于可调用对象。（对）
33. 函数和对象方法是一样的，内部实现和外部调用都没有任何区别。（错）
34. 在设计派生类时，基类的私有成员默认是不会继承的。（对）
35. 如果在设计一个类时实现类\_\_len\_\_()方法，那么该类的对象会自动支持Python内置函数len()。（对）

**第7章 文件操作**

1. 扩展库os中的方法remove()可以删除带有只读属性的文件。（错）
2. 使用内置函数open()且以”w”模式打开的文件，文件指针默认指向文件尾。（错）
3. 使用内置函数open()打开文件时，只要文件路径正确就总是可以正确打开的。（错）
4. 使用print()函数无法将信息写入文件。（错）
5. 对文件进行读写操作之后必须显式关闭文件以确保所有内容都得到保存。（对）
6. Python标准库os中的方法startfile()可以启动任何已关联应用程序的文件，并自动调用关联的程序。（对）
7. Python标准库os中的方法startfile()可以用来打开外部程序或文件，系统会自动关联相应的程序来打开或执行指定的文件。（对）
8. 假设os模块已导入，那么列表推导式 [filename for filename in os.listdir('C:\\Windows') if filename.endswith('.exe')] 的作用是列出C:\Windows文件夹中所有扩展名为.exe的文件。（对）
9. 二进制文件不能使用记事本程序打开。（错）
10. 使用普通文本编辑器软件也可以正常查看二进制文件的内容。（错）
11. 二进制文件也可以使用记事本或其他文本编辑器打开，但是一般来说无法正常查看其中的内容。（对）
12. Python标准库os中的方法isfile()可以用来测试给定的路径是否为文件。（对）
13. Python标准库os中的方法exists()可以用来测试给定路径的文件是否存在。（对）
14. Python标准库os中的方法isdir()可以用来测试给定的路径是否为文件夹。（对）
15. Python标准库os中的方法listdir()返回包含指定路径中所有文件和文件夹名称的列表。（对）
16. Python扩展库xlwt支持对Excel 2003或更低版本的Excel文件进行写操作。（对）
17. Python扩展库xlrd支持对Excel 2003或更低版本的Excel文件进行读操作。（对）
18. 标准库os的rename()方法可以实现文件移动操作。（对）
19. 标准库os的listdir()方法默认只能列出指定文件夹中当前层级的文件和文件夹列表，而不能列出其子文件夹中的文件。（对）
20. 文件对象的tell()方法用来返回文件指针的当前位置。（对）
21. 以写模式打开的文件无法进读操作。（对）
22. 假设已成功导入os和sys标准库，那么表达式 os.path.dirname(sys.executable) 的值为Python安装目录。（对）
23. 以读模式打开文件时，文件指针指向文件开始处。（对）
24. 以追加模式打开文件时，文件指针指向文件尾。（对）
25. 二进制文件也可以使用记事本程序打开，只是无法正确阅读和理解其中的内容。（对）
26. 文本文件是可以迭代的，可以使用for line in fp类似的语句遍历文件对象fp中的每一行。（对）
27. Python的主程序文件python.exe属于二进制文件。（对）
28. 对字符串信息进行编码以后，必须使用同样的或者兼容的编码格式进行解码才能还原本来的信息。（对）
29. 使用pickle进行序列化得到的二进制文件使用struct也可以正确地进行反序列化。（错）
30. 已知当前文件夹中有一个文件readme.txt具有只读属性，假设标准库os已正确导入，那么可以通过语句os.chmod('readme.txt', 0o777)来删除该文件的只读属性。（对）
31. Python标准库os的函数remove()不能删除具有只读属性的文件。（对）

**第8章 异常处理结构与程序调试、测试**

1. 在try...except...else结构中，如果try块的语句引发了异常则会执行else块中的代码。（错）
2. 异常处理结构中的finally块中代码仍然有可能出错从而再次引发异常。（对）
3. 程序中异常处理结构在大多数情况下是没必要的。（错）
4. 带有else子句的异常处理结构，如果不发生异常则执行else子句中的代码。（对）
5. 异常处理结构也不是万能的，处理异常的代码也有引发异常的可能。（对）
6. 在异常处理结构中，不论是否发生异常，finally子句中的代码总是会执行的。（对）
7. 由于异常处理结构try...except...finally...中finally里的语句块总是被执行的，所以把关闭文件的代码放到finally块里肯定是万无一失，一定能保证文件被正确关闭并且不会引发任何异常。（错）

**第9章 GUI编程**

1. 在GUI设计中，复选框往往用来实现非互斥多选的功能，多个复选框之间的选择互不影响。（对）
2. 在GUI设计中，单选按钮用来实现用户在多个选项中的互斥选择，在同一组内多个选项中只能选择一个，当选择发生变化之后，之前选中的选项自动失效。（对）

**第10章 网络程序设计**

1. Python代码可以内嵌在asp文件中。（对）
2. 无法配置IIS来支持Python程序的运行。（错）
3. 使用TCP协议进行通信时，必须首先建立连接，然后进行数据传输，最后再关闭连接。（对）
4. TCP是可以提供良好服务质量的传输层协议，所以在任何场合都应该优先考虑使用。（错）

**第11章大数据处理**

**第12章 Windows系统编程**

1. 可以使用py2exe或pyinstaller等扩展库把Python源程序打包成为exe文件，从而脱离Python环境在Windows平台上运行。（对）
2. Python程序只能在安装了Python环境的计算机上以源代码形式运行。（错）

**第13章 多线程与多进程编程**

1. 继承自threading.Thread类的派生类中不能有普通的成员方法。（错）
2. Python标准库threading中的Lock、RLock、Condition、Event、Semaphore对象都可以用来实现线程同步。（对）
3. 在编写应用程序时，应合理控制线程数量，线程并不是越多越好。（对）
4. 在多线程编程时，当某子线程的daemon属性为False时，主线程结束时会检测该子线程是否结束，如果该子线程尚未运行结束，则主线程会等待它完成后再退出。（对）
5. 在4核CPU平台上使用多线程编程技术可以很轻易地获得400%的处理速度提升。（错）
6. 多线程编程技术主要目的是为了提高计算机硬件的利用率，没有别的作用了。（错）

**第14章 数据库编程**

1. Python只能使用内置数据库SQLite，无法访问MS SQLServer、ACCESS或Oracle、MySQL等数据库。（错）

**第15章 多媒体编程**

1. 使用OpenGL画图时，画点是最基本的操作，具体生成的图形由glBegin()函数指定的mode来决定。例如，mode值为GL\_TRIANGLES时表示将要绘制三角形。（对）
2. OpenGL采用的“状态机”工作方式，一旦设置了某种状态以后，除非显式修改该状态，否则该状态将一直保持。（对）

**第16章 逆向工程与软件分析**

**第17章 科学计算与可视化**

**第18 密码学编程**

**第19章 安卓平台的Python编程**

1. 简答题
2. 简单解释Python基于值的自动内存管理方式？（Python采用的是基于值得内存管理方式，在Python中可以为不同变量赋值为相同值，这个值在内存中只有一份，多个变量指向同一个内存地址；Python具有自动内存管理功能，会自动跟踪内存中所有的值，对于没有任何变量指向的值，Python自动将其删除。）
3. 写出Python运算符&的两种功能？（1）数字位运算；2）集合交集运算。）
4. 在Python中导入模块中的对象有哪几种方式？（1)import 模块名 [as 别名]；2)from 模块名 import 对象名[ as 别名]；3)from math import \*）
5. 解释Python脚本程序的“\_\_name\_\_”变量及其作用？（每个Python脚本在运行时都有一个“\_\_name\_\_”属性。如果脚本作为模块被导入，则其“\_\_name\_\_”属性的值被自动设置为模块名；如果脚本独立运行，则其“\_\_name\_\_”属性值被自动设置为“\_\_main\_\_”。利用“\_\_name\_\_”属性即可控制Python程序的运行方式。）
6. 为什么应尽量从列表的尾部进行元素的增加与删除操作？（当列表增加或删除元素时，列表对象自动进行内存扩展或收缩，从而保证元素之间没有缝隙，但这涉及到列表元素的移动，效率较低，应尽量从列表尾部进行元素的增加与删除操作以提高处理速度。）
7. 分析逻辑运算符“or”的短路求值特性？（假设有表达式“表达式1 or 表达式2”，如果表达式1的值等价于True，那么无论表达式2的值是什么，整个表达式的值总是等价于True。因此，不需要再计算表达式2的值。）
8. 简单解释Python中短字符串驻留机制？（对于短字符串，将其赋值给多个不同的对象时，内存中只有一个副本，多个对象共享改副本。）
9. 异常和错误有什么区别？（异常是指因为程序执行过程中出错而在正常控制流以外采取的行为。严格来说，语法错误和逻辑错误不属于异常，但有些语法错误往往会导致异常，例如由于大小写拼写错误而访问不存在的对象，或者试图访问不存在的文件，等等。）
10. 使用pdb模块进行Python程序调试主要有哪几种用法？（1）在交互模式下使用pdb模块提供的功能可以直接调试语句块、表达式、函数等多种脚本。2）在程序中嵌入断点来实现调试功能。在程序中首先导入pdb模块，然后使用pdb.set\_trace()在需要的位置设置断点。如果程序中存在通过该方法调用显式插入的断点，那么在命令提示符环境下执行该程序或双击执行程序时将自动打开pdb调试环境，即使该程序当前不处于调试状态。3）使用命令行调试程序。在命令行提示符下执行“python –m pdb 脚本文件名”，则直接进入调试环境；当调试结束或程序正常结束以后，pdb将重启该程序。）
11. 阅读下面的代码，并分析假设文件“D:\test.txt”不存在的情况下两段代码可能发生的问题。

代码1：

>>> try:

fp = open(r'd:\test.txt')

print('Hello world!', file=fp)

finally:

fp.close()

代码2：

>>> try:

fp = open(r'd:\test.txt', 'a+')

print('Hello world!', file=fp)

finally:

fp.close()

答：

假设文件“D:\test.txt”不存在，那么第一段代码会抛出异常，提示fp没有定义；第二段代码执行正常。原因是第二段代码使用内置函数open()打开指定文件时如果不存在则会创建该文件，从而不会抛出异常。

1. 下面的代码本意是把当前文件夹中所有html文件都改为htm文件，仔细阅读代码，简要说明可能存在的问题。

import os

file\_list=os.listdir(".")

for filename in file\_list:

pos = filename.rindex(".")

if filename[pos+1:] == "html":

newname = filename[:pos+1]+"htm"

os.rename(filename,newname)

print(filename+"更名为："+newname)

答：

对于字符串对象，如果要查找的子字符串不存在，则rindex()方法会抛出异常。所以，如果当前文件夹中有不包含圆点的文件名或者子文件夹名，上面的代码会抛出异常而崩溃。

1. 编程题
2. 编写程序，在D盘根目录下创建一个文本文件test.txt，并向其中写入字符串hello world。

答：

fp = open(r’D:\test.txt’, ‘a+’)

print(‘hello world’, file=fp)

fp.close()

1. 写出下面代码的优化版本，提高运行效率。

x = list(range(500))

for item in x:

t = 5\*\*5

print(item+t)

答：

x = list(range(500))

t = 5\*\*5

for item in x:

print(item+t)

1. 编写程序，生成一个包含20个随机整数的列表，然后对其中偶数下标的元素进行降序排列，奇数下标的元素不变。（提示：使用切片。）

答：

import random

x = [random.randint(0,100) for i in range(20)]

print(x)

y = x[::2]

y.sort(reverse=True)

x[::2] = y

print(x)

1. 写出下面代码的执行结果。

def Join(List, sep=None):

return (sep or ',').join(List)

print(Join(['a', 'b', 'c']))

print(Join(['a', 'b', 'c'],':'))

答：

a,b,c

a:b:c

1. 写出下面代码的运行结果。

def Sum(a, b=3, c=5):

return sum([a, b, c])

print(Sum(a=8, c=2))

print(Sum(8))

print(Sum(8,2))

答：

13

16

15

1. 写出下面代码的运行结果。

def Sum(\*p):

return sum(p)

print(Sum(3, 5, 8))

print(Sum(8))

print(Sum(8, 2, 10))

答：

16

8

20

1. 编写函数，判断一个数字是否为素数，是则返回字符串YES，否则返回字符串NO。

答：

import math

def IsPrime(v):

n = int(math.sqrt(v)+1)

for i in range(2,n):

if v%i==0:

return 'No'

else:

return 'Yes'

1. 编写函数，模拟Python内置函数sorted()。

答：

def Sorted(v):

t = v[::]

r = []

while t:

tt = min(t)

r.append(tt)

t.remove(tt)

return r

1. 编写程序，生成包含20个随机数的列表，然后将前10个元素升序排列，后10个元素降序排列，并输出结果。

答：

import random

x = [random.randint(0,100) for i in range(20)]

print(x)

y = x[0:10]

y.sort()

x[0:10] = y

y = x[10:20]

y.sort(reverse=True)

x[10:20] = y

print(x)

1. 编写程序，运行后用户输入4位整数作为年份，判断其是否为闰年。如果年份能被400整除，则为闰年；如果年份能被4整除但不能被100整除也为闰年。

答：

x = input('Please input an integer of 4 digits meaning the year:')

x = eval(x)

if x%400==0 or (x%4==0 and not x%100==0):

print('Yes')

else:

print('No')

1. 编写程序，实现分段函数计算，如下表所示。

|  |  |
| --- | --- |
| x | y |
| x<0 | 0 |
| 0<=x<5 | x |
| 5<=x<10 | 3x-5 |
| 10<=x<20 | 0.5x-2 |
| 20<=x | 0 |

答：

x = input('Please input x:')

x = eval(x)

if x<0 or x>=20:

print(0)

elif 0<=x<5:

print(x)

elif 5<=x<10:

print(3\*x-5)

elif 10<=x<20:

print(0.5\*x-2)

1. 阅读下面的程序，判断其是否可以正常运行，如果可以运行则写出执行结果，如果不能运行则写出理由。

class Test:

def \_\_init\_\_(self, value):

self.\_\_value = value

@property

def value(self):

return self.\_\_value

t = Test(3)

t.value = 5

print(t.value)

答：

不能运行。程序中定义的是只读属性，不能修改属性的值。

1. 下面代码的功能是，随机生成50个介于[1,20]之间的整数，然后统计每个整数出现频率。请把缺少的代码补全。

import random

x = [random.\_\_\_\_\_\_\_\_\_\_\_\_(1,20) for i in range(\_\_\_\_\_\_\_)]

r = dict()

for i in x:

r[i] = r.get(i, \_\_\_\_\_)+1

for k, v in r.items():

print(k, v)

答：

分别填写randint、50、0

1. 假设有Python程序文件demo.py，代码如下：

def main():

if \_\_name\_\_ == '\_\_main\_\_':

print(1)

else:

print(2)

main()

将该程序文件直接运行时输出结果为\_\_\_\_\_\_\_\_\_，作为模块导入时得到结果\_\_\_\_\_\_\_\_\_\_\_-。（1、2）

1. 下面程序的执行结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1）

s = 0

for i in range(1,101):

s += i

else:

print(1)

1. 下面程序的执行结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（1275）

s = 0

for i in range(1,101):

s += i

if i == 50:

print(s)

break

else:

print(1)

1. 下面的程序是否能够正常执行，若不能，请解释原因；若能，请分析其执行结果。

from random import randint

result = set()

while True:

result.add(randint(1,10))

if len(result)==20:

break

print(result)

答：无法正确执行，因为该程序的功能是从[1,10]区间中选择20个不同的随机整数，而该区间并没有这么多整数，所以程序死循环。

1. 下面的代码是否能够正确运行，若不能请解释原因；若能，请分析其执行结果。

>>> x = list(range(20))

>>> for i in range(len(x)):

del x[i]

答：无法正确执行，因为删除列表元素时会影响其他元素在列表中的索引，上面的代码会抛出下标越界的异常。

1. 阅读下面的代码，解释其功能。

>>> x = list(range(20))

>>> for index, value in enumerate(x):

if value == 3:

x[index] = 5

答：将列表x中值为3的元素修改为5。

1. 阅读下面的代码，解释其功能。

>>> x = [range(3\*i, 3\*i+5) for i in range(2)]

>>> x = list(map(list, x))

>>> x = list(map(list, zip(\*x)))

答：首先生成一个包含列表的列表，然后模拟矩阵转置。

1. 阅读下面的代码，解释其功能。

>>> import string

>>> x = string.ascii\_letters + string.digits

>>> import random

>>> print(''.join(random.sample(x, 10)))

答：输出由英文字母大小写或数字组成的长度为10且不重复的随机字符串。

1. 阅读下面的代码，分析其执行结果。

def demo(\*p):

return sum(p)

print(demo(1,2,3,4,5))

print(demo(1,2,3))

答：输出结果为

15

6

1. 阅读下面的代码，分析其执行结果。

def demo(a, b, c=3, d=100):

return sum((a,b,c,d))

print(demo(1, 2, 3, 4))

print(demo(1, 2, d=3))

答：输出结果为

10

9

1. 下面的代码输出结果为\_\_\_\_\_\_\_\_\_\_\_\_\_。（3）

def demo():

x = 5

x = 3

demo()

print(x)

1. 下面函数的功能为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。（将序列循环左移k位，得到新序列并返回）

def demo(lst, k):

if k<len(lst):

return lst[k:]+lst[:k]

1. 编写函数，求任意整数的二进制形式中最后连续0的个数。

def demo(n):

b\_n = bin(n)

index = b\_n.rfind('1') + 1

return len(b\_n[index:])

1. 有n个乒乓球运动员打淘汰赛，编写函数计算至少需要多少场比赛才能决出冠军，不允许直接使用n-1。

def demo(n):

if n == 1:

return 0

if n == 2:

return 1

m, c = divmod(n, 2)

return m + demo(c+m)

1. 使用循环和列表推导式两种方法求解百钱买百鸡问题。假设大鸡5元一只，中鸡3元一只，小鸡1元三只，现有100元钱想买100只鸡，有多少种买法？
2. 循环

>>> for x in range(21):

for y in range(34):

z = 100-x-y

if z%3==0 and 5\*x + 3\*y + z//3 == 100:

print(x,y,z)

0 25 75

4 18 78

8 11 81

12 4 84

1. 列表推导式

>>> [(x, y, 100-x-y) for x in range(21) for y in range(34) if (100-x-y)%3==0 and 5\*x+3\*y+(100-x-y)//3==100]

[(0, 25, 75), (4, 18, 78), (8, 11, 81), (12, 4, 84)]

1. 编写函数，给定任意字符串，找出其中只出现一次的字符，如果有多个这样的字符，就全部找出。

def searchOne(s):

# 创建空字典

d = dict()

# 遍历字符串，并分别记录每个字符的出现次数

for ch in s:

# 这里重点演示字典的get()方法

# 如果这个字符出现过，加1

# 如果这个字符第一次出现，0+1

d[ch] = d.get(ch, 0) + 1

# 列表推导式，查找所有只出现一次的字符

chs = [ch for ch, n in d.items() if n==1]

# 返回最终结果，所有只出现一次的字符

return chs

print(searchOne('abcdddca'))

1. 阅读以下冒泡法排序代码，尝试写出优化代码，提高代码运行效率。

from random import randint

def bubbleSort(lst):

length = len(lst)

for i in range(0, length):

for j in range(0, length-i-1):

#比较相邻两个元素大小，并根据需要进行交换

if lst[j] > lst[j+1]:

lst[j], lst[j+1] = lst[j+1], lst[j]

lst = [randint(1, 100) for i in range(20)]

print('Before sort:\n', lst)

bubbleSort(lst)

print('After sort:\n', lst)

参考答案：

from random import randint

def bubbleSort(lst):

length = len(lst)

for i in range(0, length):

**flag = True**

for j in range(0, length-i-1):

#比较相邻两个元素大小，并根据需要进行交换

if lst[j] > lst[j+1]:

lst[j], lst[j+1] = lst[j+1], lst[j]

**flag = False**

**if flag:**

**break**

lst = [randint(1, 100) for i in range(20)]

print('Before sort:\n', lst)

bubbleSort(lst)

print('After sort:\n', lst)

1. 编写程序，用户输入带有千分位逗号的数字字符串，然后输出不带千分位逗号的数字字符串。如果输入字符串'0'则退出程序。

参考答案：

def convert(strNumber):

return ''.join(strNumber.split(','))

while True:

x = input('输入带有千分位逗号的数字：')

if x == '0':

print('bye')

break

print(convert(x))

1. 编写程序，用户输入不带千分位逗号的数字字符串，然后输出带千分位逗号的数字字符串。

参考答案：

def convert(strNumber):

# 考虑小数的情况

temp = strNumber.split('.', 1)

# 整数部分

first = temp[0]

if not first.isdigit():

return '不是有效数字'

# 小数部分

try:

second = temp[1]

if not second.isdigit():

return '不是有效数字'

except:

second = ''

# 增加千分位逗号

def nested(s):

result = []

length = len(s)

index = length % 3

if index != 0:

result.append(s[:index])

for i in range(index, length, 3):

result.append(s[i:i+3])

return ','.join(result)

first = nested(first)

# 小数部分和整数部分的千分位不一样

if second:

second = ''.join(reversed(second))

second = nested(second)

second = ''.join(reversed(second))

# 删除两侧可能的0和千分位逗号

return '.'.join([first, second]).strip(',0')

# 删除整数左侧可能的0和逗号

return first.lstrip('0,')

# 测试

while True:

x = input('输入不带千分位逗号的数字：')

if x == '0':

print('bye')

break

print(convert(x))

1. 3