附录：Python 常见面试题精选

一、 基础知识（7 题）

题01：Python 中的不可变数据类型和可变数据类型是什么意思？

难度：★☆☆☆☆

【参考答案】

不可变数据类型是指不允许变量的值发生变化，如果改变了变量的值，相当于是新建了一个对象，而对于相同的值的对象，在内存中则只有一个对象（一个地址）。数值型、字符串 string 和元组 tuple 都属于不可变数据类型。

可变数据类型是指允许变量的值发生变化，即如果对变量执行 append、+=等操作，只会改变变量的值，而不会新建一个对象，变量引用的对象的地址也不会变化。不过对于相同的值的不同对象，在内存中会存在不同的对象，即每个对象都有自己的地址，相当于内存中对于同值的对象保存了多份，这里不存在引用计数，是实实在在的对象。列表 list 和字典 dict 都属于可变数据类型。

题02：请简述 Python 中 is 和==的区别。

难度：★★☆☆☆

【参考答案】

Python 中的对象包含三个要素：id、type 和 value。is 比较的是两个对象的 id。==比较的是两个对象的 value。

题03：请简述 function(*args, **kwargs)中的 *args, **kwargs 分别是什么意思？

难度：★★☆☆☆

【参考答案】

*args 和**kwargs 主要用于函数定义的参数。Python 语言允许将不定数量的参数传给一个函数，其中*args 表示一个非键值对的可变参数列表，**kwargs 则表示不定数量的键值对参数列表。注意：*args 和**kwargs 可以同时在函数的定义中，但是*args 必须在**kwargs 前面。

题04：请简述面向对象中__new__和__init__的区别。

难度：★★★☆☆

【参考答案】

（1） __new__至少要有一个参数 cls，代表当前类，此参数在实例化时由 Python 解释器自动识别。

（2） __new__返回生成的实例，可以返回父类（通过 super(当前类名, cls)的方式）__new__出来的实例，或者直接是对象的__new__出来的实例。这在自己编程实现__new__时要特别注意。

（3） __init__有一个参数 self，就是这个__new__返回的实例，__init__在__new__的基础上可以完成一些其它初始化的动作，__init__不需要返回值。

（4） 如果__new__创建的是当前类的实例，会自动调用__init__，通过返回语句里面调用的__new__函数的第一个参数是 cls 来保证是当前类实例，如果是其他类的类名，那么实际创建并返回的就是其他类的实例，也就不会调用当前类或其他类的__init__函数。

题05：Python 子类继承自多个父类时，如多个父类有同名方法，子类将继承自哪个方法？

难度：★☆☆☆☆

【参考答案】

Python 语言中子类继承父类的方法是按照继承的父类的先后顺序确定的，例如，子类 A 继承自父类 B、C，且 B、C 中具有同名方法 Test()，那么 A 中的 Test()方法实际上是继承自 B 中的 Test()方法。

题06：请简述 Python 中如何避免死锁？

难度：★☆☆☆☆

【参考答案】

死锁是指不同线程获取了不同的锁，但是线程间又希望获取对方的锁，双方都在等待对方释放锁，这种相互等待资源的情况就是死锁。Python 语言可以使用 threading.Condition 对象，基于条件事件通知的形式去协调线程的运行，即可避免死锁。

题07：什么是排序算法的稳定性？常见的排序算法如冒泡排序、快速排序、归并排序、堆排序、Shell 排序、二叉树排序等的时间、空间复杂度和稳定性如何？

难度：★★★☆☆

【参考答案】

假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，r[i]=r[j]，且 r[i]在 r[j]之前，而在排序后的序列中，r[i]仍在 r[j]之前，则称这种排序算法是稳定的；否则称为不稳定的。

常见排序算法的时间、空间复杂度和稳定性如下表所示。

排序方法	平均时间复杂度	最好时间复杂度	最坏时间复杂度	空间复杂度	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)\sim O(n)$	不稳定
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	稳定

堆排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	不稳定
Shell 排序	$O(n\log n)$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
二叉树排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$	稳定
基数排序*	$O(d(r+n))$	$O(d(r+n))$	$O(d(r+n))$	$O(r+n)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定

*基数排序的复杂度中，r 表示关键字的基数，d 表示长度，n 表示关键字的个数。

二、 字符串与数字（7 题）

题08：字符串内容去重并按字母顺序排列。

难度：★☆☆☆☆

令 s = "hfkfdlsahfgdiuanvzx"，试对 s 去重并按字母顺序排列输出 "adfghiklnsuvxz"。

【参考答案】

```
s = "hfkfdlsahfgdiuanvzx"
s = list(set(s))
s.sort(reverse=False)
print("".join(s))
```

题09：判断两个字符串是否同构。

难度：★★☆☆☆

字符串同构是指字符串 s 中的所有字符都可以替换为 t 中的所有字符。在保留字符顺序的同时，必须用另一个字符替换所有出现的字符。不能将 s 中的两个字符映射到 t 中同一个字符，但字符可以映射到自身。试判定给定的字符串 s 和 t 是否同构。

例如：

s = "add"
t = "apple"

输出 False

s = "paper"
t = "title"

输出 True

【参考答案】

```
print(len(set(s)) == len(set(t)) == len(set(zip(s, t))))
```

题10：使用 Lambda 表达式实现将 IPv4 的地址转换为 int 型整数。

难度：★★★☆☆

例如：ip2int("192.168.3.5")

输出：3232236293

【参考答案】

```
ip2int = lambda x:sum([256**j*int(i) for j,i in enumerate(x.split('.')[::-1])])
```

题11：罗马数字使用字母表示特定的数字，试编写函数 romanToInt()，输入罗马数字字符串，输出对应的阿拉伯数字。

难度：★★★☆☆

【参考答案】

罗马数字中字母与阿拉伯数字的对应关系如下：

M:1000，CM:900，D:500，CD:400，C:100，XC:90，L:50，XL: 40，X:10，IX:9，V:5，VI:4，I:1

```
def romanToInt(s):
    table = {'M':1000, 'CM':900, 'D':500, 'CD':400, 'C':100, 'XC':90, 'L':50, 'XL': 40, 'X':10,
'IX':9, 'V':5, 'VI':4, 'I':1}
    result = 0
    for i in range(len(s)):
        if i > 0 and table[s[i]] > table[s[i-1]]:
            result += table[s[i]]
            result -= 2 * table[s[i-1]]
        else:
            result += table[s[i]]
    return result
```

题12：判断括号是否成对。

难度：★★★☆☆

给定一个只包含字符“(” “)” “{” “}” “[” 和 “]” 的字符串，试编写函数 isParenthesesValid()，输入该字符串，确定输入的字符串是否有效。括号必须以正确的顺序关闭，例如“()”和“()[]{}”都是有效的，但“(]”和“([]]”不是。

【参考答案】

```
def isParenthesesValid(s):
    pars = [None]
    parmap = {')': '(', '}': '{', ']': '['}
    for c in s:
        if c in parmap:
            if parmap[c] != pars.pop():
                return False
        else:
            pars.append(c)
    return len(pars) == 1
```

题13：编写函数输出 count-and-say 序列的第 *n* 项。

难度：★★★★☆

count-and-say 序列是一个整数序列，其前五个元素如下：

1
11
21
1211
111221

1 读作“1”或 11。11 读作“两个 1”或 21。21 读作“一个 2，然后一个 1”或 1211。即下一项是将上一项“读出来”再写成数字。
试编写函数 CountAndSay(n)，输出 count-and-say 序列的第 *n* 项。

【参考答案】

```
def CountAndSay(n):
    ans = "1"
    n -= 1
    while n > 0:
        res = ""
        pre = ans[0]
        count = 1
        for i in range(1, len(ans)):
            if pre == ans[i]:
                count += 1
            else:
                res += str(count) + pre
                pre = ans[i]
                count = 1
        res += str(count) + pre
        ans = res
```

```
        n -= 1
    return ans
```

题14：不使用 sqrt 函数，试编写 squareRoot()函数，输入一个正数，输出它的平方根的整数部分

难度：★★★★☆

【参考答案】

```
def squareRoot(x):
    result = 1.0
    while abs(result * result - x) > 0.1:
        result = (result + x / result) / 2
    return int(result)
```

三、 正则表达式（4 题）

题15：请写出匹配中国大陆手机号且结尾不是 4 和 7 的正则表达式。

难度：★☆☆☆☆

【参考答案】

```
import re
tels = ["15923456789", "14456781234", "12345678987", "11444777"]
for tel in tels:
    print("Valid") if (re.match(r"1\d{9}[0-3,5-6,8-9]", tel) != None) else print("Invalid")
```

题16：请写出以下代码的运行结果。

难度：★★☆☆☆

```
import re
str = '<div class="nam">中国</div>'
res = re.findall(r'<div class=".*">(.*?)</div>',str)
print(res)
```

结果如下：

【参考答案】

```
['中国']
```

题17：请写出以下代码的运行结果。

难度：★★★☆☆

```
import re
```

```
match = re.compile('www\....?').match("www.baidu.com")
if match:
    print(match.group())
else:
    print("NO MATCH")
```

【参考答案】

```
www.bai
```

题18：请写出以下代码的运行结果。

难度：★★☆☆☆

```
import re

example = "<div>test1</div><div>test2</div>"
Result = re.compile("<div>.*").search(example)
print("Result = %s" % Result.group())
```

【参考答案】

```
Result = <div>test1</div><div>test2</div>
```

四、 列表、字典、元组、数组、矩阵（9 题）

题19：使用递推式将矩阵转换为一维向量。

难度：★☆☆☆☆

使用递推式将

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

转换为 [1, 2, 3, 4, 5, 6]。

【参考答案】

```
a = [[1, 2], [3, 4], [5, 6]]
print([j for i in a for j in i])
```

题20：写出以下代码的运行结果。

难度：★★★★☆

```
def testFun():
```

```
    temp = [lambda x : i*x for i in range(5)]
    return temp
for everyLambda in testFun():
    print (everyLambda(3))
```

结果如下：

【参考答案】

```
12
12
12
12
12
```

题21：编写 Python 程序，打印星号金字塔。

难度：★★★☆☆

编写尽量短的 Python 程序，实现打印星号金字塔。例如 n=5 时输出以下金字塔图形：

```
    *
   ***
  *****
 *******
*********
```

参考代码如下：

【参考答案】

```
n = 5
for i in range(1,n+1):
    print(' '*(n-(i-1))+'*'*(2*i-1))
```

题22：获取数组的支配点。

难度：★★★☆☆

支配数是指数组中某个元素出现的次数大于数组元素总数的一半时就成为支配数，其所在下标称为支配点。编写 Python 函数 FindPivot(li)，输入数组，输出其中的支配点和支配数，若数组中不存在支配数，输出 None。

例如：[3,3,1,2,2,1,2,2,4,2,2,1,2,3,2,2,2,2,2,4,1,3,3]中共有 23 个元素，其中元素 2 出现了 12 次，其支配点和支配数组合是(18, 2)。

【参考答案】

```
def FindPivot(li):
    mid = len(li)/2
    for l in li:
        count = 0
        i = 0
        mark = 0
        while True:
            if l == li[i]:
                count += 1
                temp = i
            i += 1
            if count > mid:
                mark = temp
                return (mark, li[mark])
            if i > len(li) - 1:
                break
```

题23：将函数按照执行效率高低排序

难度：★★★☆☆

有如下三个函数，请将它们按照执行效率高低排序。

```
def S1(L_in):
    l1 = sorted(L_in)
    l2 = [i for i in l1 if i<0.5]
    return [i*i for i in l2]

def S2(L_in):
    l1 = [i for i in L_in if i<0.5]
    l2 = sorted(l1)
    return [i*i for i in l2]

def S3(L_in):
    l1 = [i*i for i in L_in]
    l2 = sorted(l1)
    return [i for i in l2 if i<(0.5*0.5)]
```

【参考答案】

使用 cProfile 库即可测试三个函数的执行效率：

```
import random
import cProfile

L_in = [random.random() for i in range(1000000)]
```

```
cProfile.run('S1(L_in)')
cProfile.run('S2(L_in)')
cProfile.run('S3(L_in)')
```

从结果可知，执行效率从高到低依次是 S2、S1、S3。

题24：螺旋式返回矩阵的元素

难度：★★★★★

给定 m×n 个元素的矩阵（m 行，n 列），编写 Python 函数 spiralOrder(matrix)，以螺旋顺序返回矩阵的所有元素。

例如，给定以下矩阵：

[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]

应该返回[1,2,3,6,9,8,7,4,5]

【参考答案】

```
def spiralOrder(matrix):
    if len(matrix) == 0 or len(matrix[0]) == 0:
        return []
    ans = []
    left, up, down, right = 0, 0, len(matrix) - 1, len(matrix[0]) - 1
    while left <= right and up <= down:
        for i in range(left, right + 1):
            ans += matrix[up][i],
        up += 1
        for i in range(up, down + 1):
            ans += matrix[i][right],
        right -= 1
        for i in reversed(range(left, right + 1)):
            ans += matrix[down][i],
        down -= 1
        for i in reversed(range(up, down + 1)):
            ans += matrix[i][left],
        left += 1
    return ans[:(len(matrix) * len(matrix[0]))]
```

题25：矩阵重整

难度：★★★★☆

对于一个给定的二维数组表示的矩阵，以及两个正整数 r 和 c，分别表示所需重新整形矩阵的行数和列数。reshape 函数生成一个新的矩阵，并且将原矩阵的所有元素以与原矩阵相同的行遍历顺序填充进去，将该矩阵重新整形为一个不同大小的矩阵但保留其原始数据。对于给定矩阵和参数的 reshape 操作是可以完成且合法的，则输出新的矩阵；否则，输出原始矩阵。请使用 Python 语言实现 reshape 函数。

例如：

输入	r, c	输出	说明
nums = [[1,2], [3,4]]	r = 1，c = 4	[[1,2,3,4]]	行遍历的是[1,2,3,4]。新的重新形状矩阵是 1 * 4 矩阵，使用前面的列表逐行填充。
nums = [[1,2], [3,4]]	r = 2，c = 4	[[1,2], [3,4]]	无法将 2 * 2 矩阵重新整形为 2 * 4 矩阵。所以输出原始矩阵。

注意：给定矩阵的高度和宽度在[1,100]范围内。给定的 r 和 c 都是正数。

【参考答案】

```
def matrixReshape(nums, r, c):
    """
    if r * c != len(nums) * len(nums[0]):
        return nums
    m = len(nums)
    n = len(nums[0])
    ans = [[0] * c for _ in range(r)]
    for i in range(r * c):
        ans[i / c][i % c] = nums[i / n][i % n]
    return ans
```

题26：查找矩阵中第 *k* 个最小元素。

难度：★★★★☆

给定 *n*×*n* 矩阵，其中每行每列元素均按升序排列，试编写 Python 函数 kthSmallest(matrix, k)，找到矩阵中的第 k 个最小元素。

注意：查找的是排序顺序中的第 *k* 个最小元素，而不是第 *k* 个不同元素。

例如：

矩阵=
[[1,5,9],
[10,11,13],
[12,13,15]]

k = 8，应返回 13。

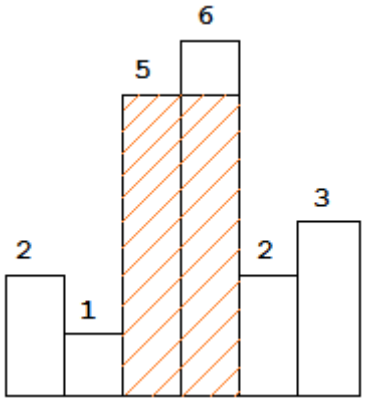
【参考答案】

```
import heapq

def kthSmallest(matrix, k):
    visited = {(0, 0)}
    heap = [(matrix[0][0], (0, 0))]

    while heap:
        val, (i, j) = heapq.heappop(heap)
        k -= 1
        if k == 0:
            return val
        if i + 1 < len(matrix) and (i + 1, j) not in visited:
            heapq.heappush(heap, (matrix[i + 1][j], (i + 1, j)))
            visited.add((i + 1, j))
        if j + 1 < len(matrix) and (i, j + 1) not in visited:
            heapq.heappush(heap, (matrix[i][j + 1], (i, j + 1)))
            visited.add((i, j + 1))
```

题27：试编写函数 largestRectangleArea()，求一幅柱状图中包含的最大矩形的面积。

难度：★★★★★

例如对于下图：



输入：[2,1,5,6,2,3]

输出：10

【参考答案】

```
def largestRectangleArea(heights):
    stack=[]
    i=0
    area=0
    while i<len(heights):
```

```
        if stack==[] or heights[i]>heights[stack[len(stack)-1]]:  # 递增直接入栈
            stack.append(i)
        else:  # 不递增开始弹栈
            curr=stack.pop()
            if stack == []:
                width = i
            else:
                width = i-stack[len(stack)-1]-1
            area=max(area,width*heights[curr])
            i-=1
        i+=1

    while stack != []:
        curr = stack.pop()
        if stack == []:
            width = i
        else:
            width = len(heights)-stack[len(stack)-1]-1
        area = max(area,width*heights[curr])
    return area
```

五、 设计模式（3 题）

题28：使用 Python 语言实现单例模式。

难度：★★★☆☆

【参考答案】

```
class Singleton(object):
    def __new__(cls, *args, **kw):
        if not hasattr(cls, '_instance'):
            orig = super(Singleton, cls)
            cls._instance = orig.__new__(cls, *args, **kw)
        return cls._instance
```

题29：使用 Python 语言实现工厂模式。

难度：★★★★☆

编写适当的 Python 程序，完成以下功能： 1. 定义基类 Person，含有获取名字，性别的方法。 2. 定义 Person 类的两个子类 Male 和 Female，含有打招呼的方法。 3. 定义工厂类，含有 getPerson 方法，接受两个输入参数：名字和性别。 4. 用户通过调用 getPerson 方法使用工厂类。

【参考答案】

```
class Person:
```

```
    def __init__(self):
        self.name = None
        self.gender = None

    def getName(self):
        return self.name

    def getGender(self):
        return self.gender

class Male(Person):
    def __init__(self, name):
        print("Hello Mr." + name)

class Female(Person):
    def __init__(self, name):
        print("Hello Miss." + name)

class Factory:
    def getPerson(self, name, gender):
        if(gender == 'M'):
            return Male(name)
        if(gender == 'F'):
            return Female(name)

if __name__ == '__main__':
    factory = Factory()
    person = factory.getPerson("Huang", "M")
```

题30：使用 Python 语言实现观察者模式。

难度：★★★★★

给定一个数字，现有的默认格式化显示程序以十进制格式显示此数字。请编写适当的 Python 程序，实现支持添加（注册）更多的格式化程序（如添加一个十六进制格式化程序和一个二进制格式化程序）。每次数值更新时，已注册的程序就会收到通知，并显示更新后的值。

【参考答案】

```
import itertools
class Publisher:
    def __init__(self):
        self.observers = set()

    def add(self, observer, *observers):
        for observer in itertools.chain((observer, ), observers):
            self.observers.add(observer)
```

```
            observer.update(self)

    def remove(self, observer):
        try:
            self.observers.discard(observer)
        except ValueError:
            print('移除 {} 失败!'.format(observer))

    def notify(self):
        [observer.update(self) for observer in self.observers]

class DefaultFormatter(Publisher):
    def __init__(self, name):
        Publisher.__init__(self)
        self.name = name
        self._data = 0

    def __str__(self):
        return "{}: '{}' 的值 = {}".format(type(self).__name__, self.name, self._data)

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, new_value):
        try:
            self._data = int(new_value)
        except ValueError as e:
            print('错误: {}'.format(e))
        else:
            self.notify()

class HexFormatter:
    def update(self, publisher):
        print("{}: '{}' 的十六进制值 = {}".format(type(self).__name__, publisher.name,
hex(publisher.data)))

class BinaryFormatter:
    def update(self, publisher):
        print("{}: '{}' 的二进制值 = {}".format(type(self).__name__, publisher.name,
bin(publisher.data)))

def main():
    df = DefaultFormatter('test1')
    print(df)
```

```
    hf = HexFormatter()
    df.add(hf)
    df.data = 37
    print(df)

    bf = BinaryFormatter()
    df.add(bf)
    df.data = 23
    print(df)

    df.remove(hf)
    df.data = 56
    print(df)

    df.remove(hf)
    df.add(bf)

    df.data = 'hello'
    print(df)

    df.data = 7.2
    print(df)

if __name__ == '__main__':
    main()
```

六、 树、二叉树、图（5 题）

题31：使用 Python 编写实现二叉树前序遍历的函数 preorder(root, res=[])。

难度：★★☆☆☆

【参考答案】

```
def preorder(root, res=[]):
    if not root:
        return
    res.append(root.val)
    preorder(root.left,res)
    preorder(root.right,res)
    return res
```

题32：使用 Python 实现一个二分查找函数。

难度：★★★☆☆

【参考答案】

```
def binary_search(num_list, x):
    num_list = sorted(num_list)
    left, right = 0, len(num_list) - 1
    while left <= right:
        mid = (left + right) // 2
        if num_list[mid] > x:
            right = mid - 1
        elif num_list[mid] < x:
            left = mid + 1
        else:
            return '待查元素{0}在排序后列表中的下标为：{1}'.format(x, mid)
    return '待查找元素%s 不存在指定列表中' %x
```

题33：编写 Python 函数 maxDepth()，实现获取二叉树 root 最大深度。

难度：★★★★☆

【参考答案】

```
def maxDepth(self, root):
        if root == None:
            return 0
        return max(self.maxDepth(root.left),self.maxDepth(root.right))+1
```

题34：输入两棵二叉树 Root1、Root2，判断 Root2 是否 Root1 的子结构（子树）。

难度：★★★★☆

【参考答案】

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def istree(pRoot1, pRoot2):
    if not pRoot2:
        return True
    if not pRoot1 and pRoot2:
        return False
    if pRoot1.val != pRoot2.val:
        return False
    elif pRoot1.val == pRoot2.val:
        return istree(pRoot1.left, pRoot2.left) and istree(pRoot1.right, pRoot2.right)
```

```
def HasSubtree(pRoot1, pRoot2):
    if not pRoot1 or not pRoot2:
        return False

    if pRoot1.val == pRoot2.val:
        return istree(pRoot1, pRoot2)
    else:
        return HasSubtree(pRoot1.left, pRoot2) or HasSubtree(pRoot1.right, pRoot2)
```

题35：判断数组是否某棵二叉搜索树后序遍历的结果。

难度：★★★★☆

编写函数 VerifySquenceOfBST(sequence)，实现以下功能：输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出 True,否则输出 False。假设输入数组的任意两个数字都互不相同。

【参考答案】

```
def VerifySquenceOfBST(sequence):
    if not sequence:
        return False
    i= 0
    for i in range(len(sequence)-1):
        if sequence[i]>sequence[-1]:
            break
    if i < len(sequence)-2:
        for j in range(i+1,len(sequence)-1):
            if sequence[j]<sequence[-1]:
                return False
    left = True
    right = True
    if i>0:
        left = VerifySquenceOfBST(sequence[:i])
    elif i< len(sequence)-2:
        right = VerifySquenceOfBST(sequence[i:-1])
    return left and right
```

七、 文件操作（3 题）

题36：计算 test.txt 中的大写字母数。

难度：★☆☆☆☆

【参考答案】

```
import os
os.chdir('D:\\')
with open('test.txt') as test:
    count = 0
    for i in test.read():
        if i.isupper():
            count+=1
    print(count)
```

题37：补全缺失的代码。

难度：★★☆☆☆

```
def print_directory_contents(sPath):
    # 补充该函数的实现代码
```

print_directory_contents()函数接受文件夹路径名称作为输入参数，返回其中包含的所有子文件夹和文件的完整路径。

【参考答案】

```
def print_directory_contents(sPath):
    import os
    for sChild in os.listdir(sPath):
        sChildPath = os.path.join(sPath,sChild)
        if os.path.isdir(sChildPath):
            print_directory_contents(sChildPath)
        else:
            print(sChildPath)
```

题38：设计内存中的文件系统。

难度：★★★★☆

使用 Python 语言设计内存中的文件系统，实现以下命令。

ls：给定字符串格式的路径。如果是文件路径，则返回仅包含此文件名称的列表。如果是目录路径，则返回此目录中的文件和目录名称列表。输出结果（文件和目录名称）应按字典顺序排列。

mkdir：如果目录路径不存在，则应根据路径创建新目录。如果路径中的中间目录也不存在，则也应该创建它们。此函数具有 void 返回类型。

注：

可以假设所有文件或目录路径都是以/开头并且不以/结尾的绝对路径，除了路径只是“/”。

可以假设所有操作都将传递有效参数，用户不会尝试检索文件内容或列出不存在的目录或文件。

可以假设所有目录名称和文件名只包含小写字母，并且同一目录中不存在相同的名称。

【参考答案】

```
class FileNode(object):
    def __init__(self, name):
        self.isFolder = True
        self.childs = {}
        self.name = name
        self.data = ""

class FileSystem(object):
    def __init__(self):
        self.root = FileNode("/")

    def ls(self, path):
        fd = self.lookup(path, False)
        if not fd:
            return []
        if not fd.isFolder:
            return [fd.name]
        files = []
        for file in fd.childs:
            files.append(file)
        files.sort()
        return files

    def lookup(self, path, isAutoCreate):
        path = path.split("/")
        p = self.root
        for name in path:
            if not name:
                continue
            if name not in p.childs:
                if isAutoCreate:
                    p.childs[name] = FileNode(name)
                else:
                    return None
            p = p.childs[name]
        return p

    def mkdir(self, path):
        self.lookup(path, True)

# 测试
obj = FileSystem()
obj.mkdir("/test/path")
obj.ls("/test")
```

八、 网络编程（4 题）

题39：请至少说出三条 TCP 和 UDP 协议的区别。

难度：★★☆☆☆

【参考答案】

（1） TCP 面向连接（如打电话要先拨号建立连接）；UDP 是无连接的，即发送数据之前不需要建立连接。
（2） TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达；UDP 尽最大努力交付，即不保证可靠交付。
（3） TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流；UDP 是面向报文的，UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）。
（4） 每一条 TCP 连接只能是点到点的；UDP 支持一对一，一对多，多对一和多对多的交互通信。
（5） TCP 首部开销 20 字节；UDP 的首部开销小，只有 8 个字节。
（6） TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道。

题40：请简述 Cookie 和 Session 的区别。

难度：★☆☆☆☆

【参考答案】

（1） Session 在服务器端，Cookie 在客户端（浏览器）。
（2） Session 可以存放在文件、数据库或内存中，默认以文件方式保存。
（3） Session 的运行依赖 Session ID，而 Session ID 保存在 Cookie 中。因此，如果浏览器禁用了 Cookie，同时 Session 也会失效。

题41：请简述向服务器端发送请求时的 GET 方式与 POST 方式的区别。

难度：★☆☆☆☆

【参考答案】

（1） 在浏览器回退时 GET 方式没有变化，而 POST 会再次提交请求。
（2） GET 请求会被浏览器主动缓存，而 POST 不会，除非手动设置。
（3） GET 请求只能进行 URL 编码，而 POST 支持多种编码方式。
（4） GET 请求参数会被完整保留在浏览器历史记录里，而 POST 中的参数不会被保留。
（5） 对参数的数据类型，GET 只接受 ASCII 字符，而 POST 没有限制。
（6） GET 比 POST 更不安全，因为参数直接暴露在 URL 上，所以不能用来传递敏感信息。
（7） GET 参数通过 URL 传递且传送的参数是有长度限制的，POST 放在 Request body 中且长度没有限制。
（8） 对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）；而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

题42：使用 threading 组件编写支持多线程的 Socket 服务端。

难度：★★★★☆

使用 Python 语言的 threading 组件编写支持多线程的 Socket 服务端，支持-x 和-p 参数，分别表示指定最大连接数和监听端口。

【参考答案】

```
import socket
import threading,getopt,sys,string

opts, args = getopt.getopt(sys.argv[1:], "hp:l:",["help","port=","list="])

list=50
port=8001

for op, value in opts:
    if op in ("-x","--maxconn"):
        list = int(value)
    elif op in ("-p","--port"):
        port = int(value)

def Config(client, address):
    try:
        client.settimeout(500)
        buf = client.recv(2048)
        client.send(buf)
    except socket.timeout:
        print('time out')
    client.close()

def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost', port))
    sock.listen(list)
    while True:
        client,address = sock.accept()
        thread = threading.Thread(target=Config, args=(client, address))
        thread.start()

if __name__ == '__main__':
    main()
```

九、 数据库编程（6 题）

题43：简述数据库的第一、第二、第三范式的内容。

难度：★★☆☆☆

【参考答案】

范式是“符合某一种级别的关系模式的集合，表示一个关系内部各属性之间的联系的合理化程度”，实际上就是一张数据表的表结构所符合的某种设计标准的级别。

1NF 的定义为：符合 1NF 的关系中的每个属性都不可再分。1NF 是所有关系型数据库的最基本要求。

2NF 在 1NF 的基础之上，消除了非主属性对于码的部分函数依赖。判断步骤如下。

第一步：找出数据表中所有的码。

第二步：根据第一步所得到的码，找出所有的主属性。

第三步：数据表中，除去所有的主属性，剩下的就都是非主属性了。

第四步：查看是否存在非主属性对码的部分函数依赖。

3NF 在 2NF 的基础之上，消除了非主属性对于码的传递函数依赖。

题44：根据以下数据表结构和数据，按照要求编写 SQL 语句。

难度：★★☆☆☆

数据库中现有 Course、Score、Student 和 Teacher 四张数据表，其中数据分别如下所示。

```
1  select * from Course
```

信息 结果 1

c_id	c_name	t_id
01	语文	02
02	数学	01
03	英语	03

```
1  select * from Score
```

信息 结果 1

s_id	c_id	s_score
01	01	80
01	02	90
01	03	99
02	01	70
02	02	60
02	03	80
03	01	80
03	02	80
03	03	80
04	01	50
04	02	30
04	03	20
05	01	76
05	02	87
06	01	31
06	03	34
07	02	89
07	03	98

```
1  select * from Student
```

信息 结果 1

s_id	s_name	s_birth	s_sex
01	赵雷	1990-01-01	男
02	钱电	1990-12-21	男
03	孙风	1990-05-20	男
04	李云	1990-08-06	男
05	周梅	1991-12-01	女
06	吴兰	1992-03-01	女
07	郑竹	1989-07-01	女
08	王菊	1990-01-20	女

```
1  select * from Teacher
```

信息 结果 1

t_id	t_name
01	张三
02	李四
03	王五

试编写 SQL 语句，查询平均成绩大于 80 的所有学生的学号、姓名和平均成绩。

【参考答案（使用 SQLServer）】

```
select a.s_id, b.s_name, avg(a.s_score) as avg_score
from Score a, Student b
where a.s_id=b.s_id
group by a.s_id, b.s_name
having avg(a.s_score)>80
```

查询结果如下：

```
1  select a.s_id, b.s_name, avg(a.s_score) as avg_score
2  from Score a, Student b
3  where a.s_id=b.s_id
4  group by a.s_id, b.s_name
5  having avg(a.s_score)>80
```

信息 结果 1

s_id	s_name	avg_score
01	赵雷	89
05	周梅	81
07	郑竹	93

题45：按照 44 题所给条件，编写 SQL 语句查询没有学全所有课程的学生信息。

难度：★★☆☆☆

【参考答案（使用 SQLServer）】

```
select b.s_id, b.s_name
from Score a, Student b
where a.s_id=b.s_id
```

```
group by b.s_id, b.s_name
having count(a.c_id)<(select count(c_id) from Course)
```

查询结果如下：

```
select b.s_id, b.s_name
from Score a, Student b
where a.s_id=b.s_id
group by b.s_id, b.s_name
having count(a.c_id)<(select count(c_id) from Course)
```

信息 结果 1

s_id	s_name
05	周梅
06	吴兰
07	郑竹

题46：按照 44 题所给条件，编写 SQL 语句查询所有课程第 2 名和第 3 名的学生信息及该课程成绩。

难度：★★★★☆

【参考答案（使用 SQLServer）】

```
select * from (
select a.s_id, b.s_name, b.s_birth, b.s_sex, a.c_id, a.s_score,
row_number() over (partition by a.c_id order by a.s_score desc) rno
from Score a, Student b
where a.s_id=b.s_id
) as a where a.rno in (2, 3) order by c_id, rno
```

查询结果如下：

```
select * from (
select a.s_id, b.s_name, b.s_birth, b.s_sex, a.c_id, a.s_score,
row_number() over (partition by a.c_id order by a.s_score desc) rno
from Score a, Student b
where a.s_id=b.s_id
) as a where a.rno in (2, 3) order by c_id, rno
```

信息 结果 1

s_id	s_name	s_birth	s_sex	c_id	s_score	rno
03	孙风	1990-05-20	男	01	80	2
05	周梅	1991-12-01	女	01	76	3
07	郑竹	1989-07-01	女	02	89	2
05	周梅	1991-12-01	女	02	87	3
07	郑竹	1989-07-01	女	03	98	2
02	钱电	1990-12-21	男	03	80	3

题47：按照 44 题所给条件，编写 SQL 语句查询所教课程有 2 人及以上不及格的教师、课程、学生信息及该课程成绩。

难度：★★★☆☆

【参考答案（使用 SQLServer）】

```
select a.s_id, d.s_name, a.s_score, a.c_id, b.c_name, b.t_id, c.t_name
from Score a, Course b, Teacher c, Student d
where a.c_id=b.c_id and b.t_id=c.t_id and a.s_id=d.s_id
and a.s_score<60 and c.t_id in (
         SELECT c.t_id
         FROM Score a, Course b, Teacher c
         where a.c_id=b.c_id and b.t_id=c.t_id and a.s_score<60
         group by c.t_id
         having count(1)>1
)
```

查询结果如下：

```
1  select a.s_id, d.s_name, a.s_score, a.c_id, b.c_name, b.t_id, c.t_name
2  from Score a, Course b, Teacher c, Student d
3  where a.c_id=b.c_id and b.t_id=c.t_id and a.s_id=d.s_id
4  and a.s_score<60 and c.t_id in (
5       SELECT c.t_id
6       FROM Score a, Course b, Teacher c
7       where a.c_id=b.c_id and b.t_id=c.t_id and a.s_score<60
8       group by c.t_id
9       having count(1)>1
10 )
```

信息 结果 1

s_id	s_name	s_score	c_id	c_name	t_id	t_name
04	李云	50	01	语文	02	李四
04	李云	20	03	英语	03	王五
06	吴兰	31	01	语文	02	李四
06	吴兰	34	03	英语	03	王五

题48：按照 44 题所给条件，编写 SQL 语句生成每门课程的一分段表（课程 ID、课程名称、分数、该课程该分数人数、该课程累计人数）。

难度：★★★★☆

【参考答案（使用 SQLServer）】

```
select a.c_id, b.c_name, a.s_score, COUNT(1) 人数,
(select COUNT(1) from Score c where c.c_id=a.c_id and c.s_score>=a.s_score) 累计人数
from Score a, Course b
where a.c_id=b.c_id
group by a.c_id, b.c_name, a.s_score
order by a.c_id, b.c_name, a.s_score desc;
```

查询结果如下：

```sql
select a.c_id, b.c_name, a.s_score, COUNT(1) 人数,
(select COUNT(1) from Score c where c.c_id=a.c_id and c.s_score>=a.s_score) 累计人数
from Score a, Course b
where a.c_id=b.c_id
group by a.c_id, b.c_name, a.s_score
order by a.c_id, b.c_name, a.s_score desc;
```

信息 结果 1

c_id	c_name	s_score	人数	累计人数
01	语文	80	2	2
01	语文	76	1	3
01	语文	70	1	4
01	语文	50	1	5
01	语文	31	1	6
02	数学	90	1	1
02	数学	89	1	2
02	数学	87	1	3
02	数学	80	1	4
02	数学	60	1	5
02	数学	30	1	6
03	英语	99	1	1
03	英语	98	1	2
03	英语	80	2	4
03	英语	34	1	5
03	英语	20	1	6

十、 图形图像与可视化（2 题）

题49：绘制一个二次函数的图形，并同时画出使用梯形法求积分时的各个梯形。

难度：★★★☆☆

【参考答案】

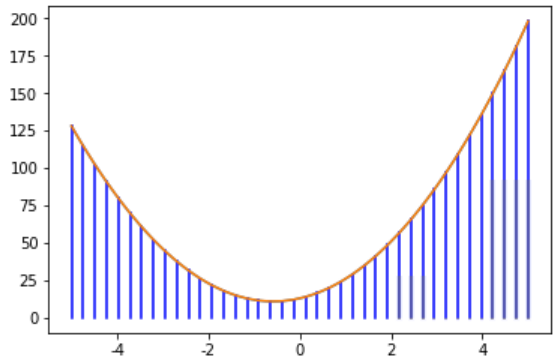
以$f(x)=6x^2+7x+13$为例，代码如下：

```
def Quadratic_Function(x):
    return 6*x**2 + 7*x + 13

import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact

def plot_ladder(laddernum):
    x = np.linspace(-5, 5, num=100)
    y = Quadratic_Function(x)
    plt.plot(x, y)

    a = np.linspace(-5, 5, num=laddernum)
    for i in range(laddernum):
        plt.plot([a[i], a[i]], [0, Quadratic_Function(a[i])], color="blue")

    ladders = []
```

```
    for i in range(laddernum):
        ladders.append([a[i], Quadratic_Function(a[i])])

    npladders = np.array(ladders)
    plt.plot(npladders[:,0], npladders[:,1])

interact(plot_ladder, laddernum=(1, 80, 1))
```

运行结果如下（使用 Jupyter Notebook）：

```
In [5]: def Quadratic_Function(x):
            return 6*x**2 + 7*x + 13

        import numpy as np
        import matplotlib.pyplot as plt
        from ipywidgets import interact

        def plot_ladder(laddernum):
            x = np.linspace(-5, 5, num=100)
            y = Quadratic_Function(x)
            plt.plot(x, y)

            a = np.linspace(-5, 5, num=laddernum)
            for i in range(laddernum):
                plt.plot([a[i], a[i]], [0, Quadratic_Function(a[i])], color="blue")

            ladders = []
            for i in range(laddernum):
                ladders.append([a[i], Quadratic_Function(a[i])])

            npladders = np.array(ladders)
            plt.plot(npladders[:,0], npladders[:,1])

        interact(plot_ladder, laddernum=(1, 80, 1))
```




```
Out[5]: <function __main__.plot_ladder(laddernum)>
```

题50：将给定数据可视化并给出分析结论

难度：★★★☆☆

某门店部分顾客的年龄、月收入以及每月平均在本店消费情况如下表所示。

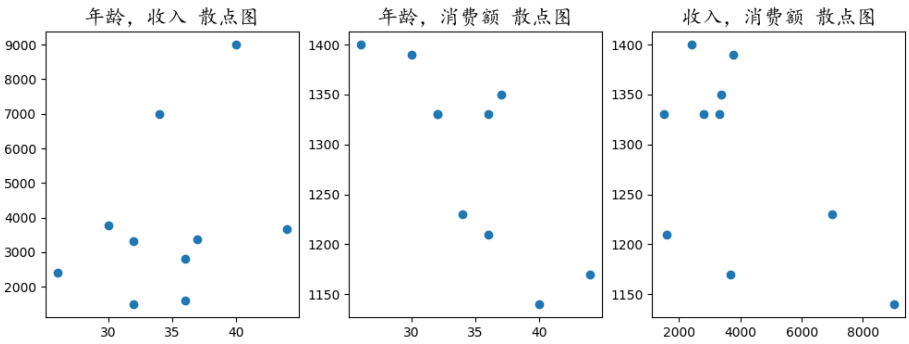
年龄	收入	每月平均在本店消费额
34	7000	1230

年龄	收入	每月平均在本店消费额
40	9000	1140
37	3380	1350
30	3780	1390
44	3660	1170
36	1600	1210
32	3320	1330
26	2400	1400
32	1500	1330
36	2800	1330

请据此使用 Python 语言绘制适当的图形并分析得到结论。

【参考答案】

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
# 年龄
age = [34, 40, 37, 30, 44, 36, 32, 26, 32, 36]
# 收入
income = [7000, 9000, 3380, 3780, 3660, 1600, 3320, 2400, 1500, 2800]
# 消费额
expense = [1230, 1140, 1350, 1390, 1170, 1210, 1330, 1400, 1330, 1330]
# 年龄，收入 散点图
ax1 = plt.subplot(121)
ax1.scatter(age, income)
ax1.set_title('年龄，收入 散点图', family='kaiti', size=16)
# 年龄，消费额 散点图
ax2 = plt.subplot(122)
ax2.scatter(age, expense)
ax2.set_title('年龄，消费额 散点图', family='kaiti', size=16)
plt.show()
```



从图中可以看到，顾客年龄与消费额几近负相关，收入与消费额也几乎负相关，而年龄与收入之间没有特别明显的关联关系。