CCF 编程能力等级认证
Grade Examination of Software Programming

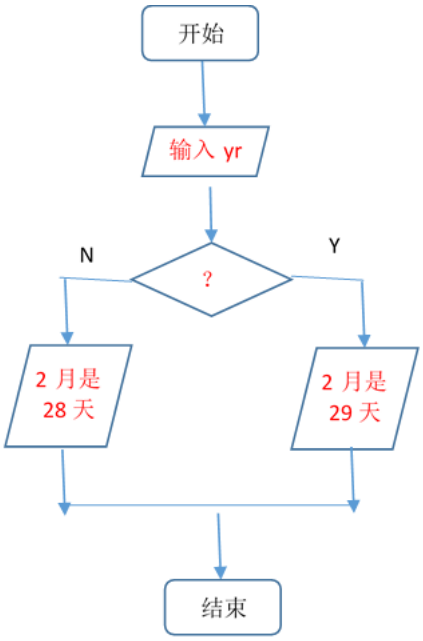
Python　五级

2024 年 03 月

1　单选题（每题 2 分，共 30 分）

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	B	C	B	D	D	A	D	C	D	A	B	A	B	B	C

第 1 题 下面流程图在yr输入2024时，可以判定yr代表闰年，并输出 2月是29天 ，则图中菱形框中应该填入（ ）。



- A. `(yr % 400 == 0)  or  (yr % 4 == 0)`
- B. `( yr % 400 == 0)  or  (yr % 4 == 0 and  yr %100 != 0)`
- C. `( yr % 400 == 0)  and  ( yr % 4 == 0)`
- D. `(yr % 400 == 0) and (yr % 4 == 0 and yr % 100 != 0)`

第 2 题 在TCP协议中，完成连接建立需要通过（ ）握手。

- A. 一次
- B. 二次
- C. 三次
- D. 四次

第 3 题 下面有关排序算法的说法，正确的是（ ）

- A. 快速排序是稳定排序

☐ **B.** Python中list类型的sort()是稳定排序

☐ **C.** 冒泡排序是不稳定排序

☐ **D.** 归并排序是不稳定排序

第 4 题 不同的排序算法，其空间复杂度也不同。与冒泡法排序空间复杂度相同的是（ ）

☐ **A.** 归并排序

☐ **B.** 快速排序

☐ **C.** 计数排序

☐ **D.** 插入排序

第 5 题 下面Python代码中，aFactorial()和bFactorial()用于求正整数的阶乘。有关说法，错误的是(　　)。

```
def aFactorial(N):
    rst = 1
    for i in range(1,N + 1):
        rst *= i
    return rst

def bFactorial(N):
    if N == 1 or N == 0:
        return 1
    return N * bFactorial(N-1)

print(aFactorial(10),bFactorial(10))
```

☐ **A.** aFactorial( )用循环方式，bFactorial()递归方式

☐ **B.** bFactorial( )更加符合数学定义，直观易于理解，而aFactorial()需要将数学定义转换为计算机程序实现

☐ **C.** 当N值较大时，aFactorial( )执行效率更高，而bFactorial()因为有多次函数调用，效率将降低，且N如果较大，将可能导致不能使用

☐ **D.** bFactorial( )因为代码量较少，没有循环，因此其执行效率更高

第 6 题 有关下面Python代码的说法，正确的是(　　)。

```
def qSort(lst):
    if len(lst) <= 1:
        return lst
    else:
        Pivot = lst[0]
        Less = [x for x in lst[1:] if x <= Pivot]
        Greater = [x for x in lst[1:] if x > Pivot]
        return qSort(Less) + [Pivot] + qSort(Greater)

lstA = [1,2,11,12,21,21,2,3,41,4,3]
lstB = qSort(lstA)
print(lstA,lstB)
```

☐ **A.** 代码中qSort()函数不是稳定排序

☐ **B.** 代码中qSort()函数空间复杂度为O(1)

☐ **C.** 代码中qSort()函数是就地排序

☐ **D.** 代码中qSort()函数是外排序，因为排序后的结果保存在新的内存空间即外空间

第 7 题 上题不能支持其他常见类型的排序，如实现该支持，横线处分别应填写代码是( )。

```
def qSort(iterData):

    if ____________________:
        ____________________
    else:
        lst = iterData

    if len(lst) <= 1:
        return lst
    else:
        Pivot = lst[0]
        Less = [x for x in lst[1:] if x <= Pivot]
        Greater = [x for x in lst[1:] if x > Pivot]
        return qSort(Less) + [Pivot] + qSort(Greater)

tplA = (1,2,11,12,21,21,2,3,41,4,3)

lstB = qSort(tplA)
print(tplA,lstB)
```

- ☐ **A.** `isinstance(iterData, list) == False`， `st == [x for x in iterData]`
- ☐ **B.** `type(iterData) == list`， `lst = [x for x in iterData]`
- ☐ **C.** `isinstance(iterData, list)`， `lst = list(iterData)`
- ☐ **D.** `type(iterData) != list`， `lst = list(iterData)`

第 8 题 上上题qSort()函数不支持排序规则函数，形如sorted()函数的key参数，为实现类似目标，横线处分别应填入代码是(　)。

```
def qSort(lst,fx = None):

    if len(lst) <= 1:
        return lst
    else:
        Pivot = lst[0]
        if ________:
            Less = [x for x in lst[1:] if _________]
            Greater = [x for x in lst[1:] if _____________]
            return qSort(Less,fx) + [Pivot] + qSort(Greater,fx)
        else:
            Less = [x for x in lst[1:] if x <= Pivot]
            Greater = [x for x in lst[1:] if x > Pivot]
            return qSort(Less,fx) + [Pivot] + qSort(Greater,fx)

lstA = [1,2,11,12,21,21,2,3,41,4,3]

lstB = qSort(lstA, lambda x:x % 10)
print(lstA, lstB)
```

- ☐ **A.** `fx == None`， `fx(x) >= fx(Pivot)`， `fx(x) < fx(Pivot)`
- ☐ **B.** `fx == None`， `fx(x) >= Pivot`， `fx(x) < Pivot`
- ☐ **C.** `fx != None`， `fx(x) >= fx(Pivot)`， `fx(x) < fx(Pivot)`
- ☐ **D.** `fx != None`， `fx(x) >= Pivot`， `fx(x) < Pivot`

第 9 题 下面的Python代码中merge()函数的两个参数均为list类型，且是已按相同规则排序的数据。下面有关说法中，正确的是（　）。

```
1  def merge(arr1, arr2):
2      result = []
3      while arr1 and arr2:
4          if arr1[0] < arr2[0]:
5              result.append(arr1.pop(0))
6          else:
7              result.append(arr2.pop(0))
8      if arr1:
9          result += arr1
10     if arr2:
11         result += arr2
12     return result
```

- [ ] **A.** 第3-7行代码将导致死循环，因为没有循环变量及其改变

- [ ] **B.** 第5行和第7行代码执行后，result的成员值为None

- [ ] **C.** 第9行和第11行是否被执行，与arr1和arr2的成员值有关，如果值转换为False，将不会被执行

- [ ] **D.** merge()函数的代码没有错误，执行后参数arr1和arr2将合并成新的list保存到result之中，且有序

第 10 题 阅读下面Python代码，横线处应填入（　）。

```
1  def Check(N,Fx):
2      return Fx(N)
3  def isOdd(N):
4      return "偶数" if N % 2 == 0 else "奇数"
5
6  isEven = lambda N:"偶数" if N % 2 == 0 else "奇数"
7
8  print(Check(10,________),Check(11,____________))
```

- [ ] **A.** `isOdd` ， `isEven`

- [ ] **B.** `isOdd` ， `isEven(10)`

- [ ] **C.** `isOdd(10)` ， `isEven`

- [ ] **D.** `isOdd(10)` ， `isEven(10)`

第 11 题 下面Python代码的平均时间复杂度是（　）。

```
1  def gcd(N,M):
2      return N if M == 0 else gcd(M, N % M)
```

- [ ] **A.** $O(N)$

- [ ] **B.** $O(\log N)$

- [ ] **C.** $O(N \log N)$

- [ ] **D.** $O(N^2)$

第 12 题 有关下面Python代码的说法，正确的是（　）。

```
1  def bSearch(lst, val):
2      def __bSearch(lst, Low, High, queryVal):
3          if Low > High:
4              return -1
5
6          midIdx = (Low + High) // 2
7          midVal = lst[midIdx]
8
9          if queryVal == midVal:
10             return midIdx
11         elif queryVal < midVal:
12             return __bSearch(lst, Low, midIdx - 1, queryVal)
13         else:
14             return __bSearch(lst, midIdx + 1, High, queryVal)
15
16     return __bSearch(lst, 0, len(lst), val)
17
18 lst = list(range(10))
19 print(bSearch(lst,3))
```

- [ ] **A.** 代码采用二分法查找，仅对有序list有效，不适用于set、dict等
- [ ] **B.** 在函数内定义函数，存在多次调用多次定义，因此存在错误
- [ ] **C.** 第16行代码__bSearch()最后一个参数val应为queryVal
- [ ] **D.** 第16行代码应为return __bSearch

第 13 题 在上题的算法中，其时间复杂度是（ ）。

- [ ] **A.** $O(N)$
- [ ] **B.** $O(\log N)$
- [ ] **C.** $O(N \log N)$
- [ ] **D.** $O(N^2)$

第 14 题 下面的Python代码中，idList变量为list类型，保存有大量身份编码，倒数第二位如为偶数，则为女性，奇数为男性，从第7位开始连续8位为出生年月日，年为4位数月和日均为两位数。编码统计每天有几位男生生日女生生日。横线处应填入代码是（ ）。

```
1 dictGender_Birthday = {} #性别，出生月日的字典
2 for everyID in idList:
3     gender, Key = int(everyID[-2]) % 2, everyID[10:14]
4     male,female = _____________________
5     lastData = dictGender_Birthday.get(Key, _____________________ )
6     dictGender_Birthday[Key] =  ( lastData[0] + male, lastData[1] + female )
7 listGender_Birthday = sorted(dictGender_Birthday.items())
8 print(listGender_Birthday)
```

- [ ] **A.** `(1, 0) if gender == 1 else (1, 0)` , `(0, 1)`
- [ ] **B.** `(0, 1) if gender == 1 else (1, 0)` , `(0, 0)`
- [ ] **C.** `0, 1 if gender == 1 else 1, 0` , `0, 0`
- [ ] **D.** `(0, 1) if gender == 1 else (1, 0)` , `0, 0`

第 15 题 有关下面Python代码的说法错误的是（ ）。

```python
class Node:
    def __init__(self, Val, Prv = None, Nxt = None):
        self.Value = Val
        self.Previous, self.Next = Prv, Nxt
    def setPrevious(self, Prv = None):
        self.Previous = Prv
    def setNext(self, Nxt = None):
        self.Next = Nxt

firstNode = Node(1)
firstNode.Next = Node(2,firstNode)
firstNode.Next.Next = Node(3,firstNode.Next)
firstNode.Next.setPrevious(firstNode)
firstNode.Next.Next.setPrevious(firstNode.Next)

secondNode = firstNode.Next
firstNode.Next = secondNode.Next
secondNode.Next.Previous = firstNode
```

- ☐ **A.** 代码中第17行执行后，firstNode(第一个节点)的下一个节点指向第3个节点，即secondNode(第2个节点)的下一个
- ☐ **B.** 代码中第18行执行后，第3个节点的Previous(前向)指向第1个节点（firstNode）
- ☐ **C.** 仅仅通过firstNode节点，不能访问第2个节点，第2个节点已不在内存中存在，自动释放所占内存
- ☐ **D.** 在第18行后，执行del secondNode后，第2个节点所占内存才会被释放。仅仅执行先有第16-18行，第2个节点内存不会被释放。

2 判断题（每题 2 分，共 20 分）

题号	1	2	3	4	5	6	7	8	9	10
答案	×	×	√	√	√	×	×	√	√	×

第 1 题 一个算法设计合理的话是可以没有输出的，比如冒泡排序就不输出任何信息（ ）。

第 2 题 流程图描述算法的能力是有限的，比如它无法对能够提前终止的循环给出等价描述 ( )。

第 3 题 归并排序的空间复杂度是$O(N)$。( )

第 4 题 在Python中，当对dict类型进行in运算查找元素是否存在时，其时间复杂度为 `O(1)` ，set类型也如此。（ ）

第 5 题 在以下Python代码中，执行后输出是 `5=>4=>3=>2=>1=>2=>3=>2=>1=>5` 。( )

```python
def Fibo(N):
    print(N,end="=>")
    if N == 1 or N == 2:
        return 1
    else:
        return Fibo(N - 1) + Fibo(N - 2)

print(Fibo(5))
```

第 6 题 贪心算法虽然可能不是局部最优，但可达到全局最优。( )

第 7 题 Python内置函数sorted()可对支持for-in循环的数据类型排序。（ ）

第 8 题 冒泡排序是就地排序，空间复杂度为 `O(1)` 。（ ）

第 9 题 下面的Python代码能实现N的质因数分解，类似埃氏筛法。（ ）

```python
def Factorization(N):
    rst = f"{N}="
    i = 2
    while N != 1:
        if N % i == 0:
            rst += f"{i}*"
            N //= i
        else:
            i += 1
    return rst[:-1]

print(Factorization(45))
```

第 10 题 Python代码print(sorted(list(range(10,20)), key = hex))执行后将输出[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]。（ ）

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：成绩排序

3.1.1 问题描述

有 N 名同学，每名同学有语文、数学、英语三科成绩。你需要按如下规则对所有同学的成绩从高到低排序：

1. 比较总分，高者靠前；
2. 如果总分相同，则比较语文和数学两科总分，高者靠前；
3. 如果仍相同，则比较语文和数学两科的最高分，高者靠前；
4. 如果仍相同，则二人并列。

你需要输出每位同学的排名，如遇 x 人并列，则他们排名相同，并留空后面的 $x-1$ 个名次。例如，有 3 名同学并列第 1，则后一名同学自动成为第 4 名。

3.1.2 输入描述

第一行一个整数 N，表示同学的人数。

接下来 N 行，每行三个非负整数 c_i, m_i, e_i 分别表示该名同学的语文、数学、英语成绩。

保证 $0 \leq c_i, m_i, e_i \leq 150$。

3.1.3 输出描述

输出 N 行，按输入同学的顺序，输出他们的排名。

注意：请不要按排名输出同学的序号，而是按同学的顺序输出他们各自的排名

3.1.4 特别提醒

在常规程序中，输入、输出时提供提示是好习惯。但在本场考试中，由于系统限定，请不要在输入、输出中附带任何提示信息。

3.1.5 样例输入 1

```
6
140 140 150
140 149 140
148 141 140
141 148 140
145 145 139
0 0 0
```

3.1.6 样例输出 1

```
1
3
4
4
2
6
```

3.1.7 数据规模

对于 30 的测试点，保证 $N \leq 100$，且所有同学的总分各不相同。

对于所有测试点，保证 $2 \leq N \leq 10^4$。

3.1.8 参考程序

```python
N = int(input())
students = []
for i in range(N):
    c, m, e = list(map(int, input().split()))
    students.append((c + m + e, c + m, max(c, m), i))

students.sort(reverse = True)

rank = [None for i in range(N)]
last_student = None
for i, student in enumerate(students):
    if student[:-1] != last_student:
        last_student = student[:-1]
        curr_rank = i + 1
    rank[student[-1]] = curr_rank

for r in rank:
    print(r)
```

3.2 编程题 2

- **试题名称**：B-smooth 数

3.2.1 题面描述

小杨同学想寻找一种名为 B-smooth 数的正整数。

如果一个正整数的最大质因子不超过 B，则该正整数为 B-smooth 数。

小杨同学想知道，对于给定的 n 和 B，有多少个不超过 n 的 B-smooth 数。

3.2.2 输入格式

第一行包含两个正整数 n, B，含义如题面所示。

3.2.3 输出格式

输出一个非负整数，表示不超过 n 的 B-smooth 数的数量。

3.2.4 样例1

```
10 3
```

```
7
```

3.2.5 样例解释

在不超过 10 的正整数中，3-smooth 数有 $\{1, 2, 3, 4, 6, 8, 9\}$，共 7 个。

3.2.6 数据范围

子任务编号	数据点占比	n	B
1	30	≤ 1000	$1 \leq B \leq 1000$
2	30	$\leq 10^6$	$\sqrt{n} \leq B \leq 10^6$
3	40	$\leq 10^6$	$1 \leq B \leq 10^6$

对于全部数据，保证有 $1 \leq n \leq 10^6$，$1 \leq B \leq 10^6$。

3.2.7 参考程序

```python
n, B = map(int, input().split())

is_prime = [True] * (n + 1)
primes = []
max_prime_factor = [0] * (n + 1)

for i in range(2, n + 1):
    if is_prime[i]:
        primes.append(i)
        max_prime_factor[i] = i
    for p in primes:
        if i * p > n:
            break
        is_prime[i * p] = False
        max_prime_factor[i * p] = max(max_prime_factor[i], p)
        if i % p == 0:
            break

cnt = 0
for i in range(1, n + 1):
```

```
    if max_prime_factor[i] <= B:
        cnt += 1
print(cnt)
```