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

Python　五级

2023 年 9 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	A	D	C	C	A	B	A	C	D	D	B	C	B	A	C

第 1 题 近年来，线上授课变得普遍，很多有助于改善教学效果的设备也逐渐流行，其中包括比较常用的手写板，那么它属于哪类设备？（ ）。

- ☐ **A.** 输入
- ☐ **B.** 输出
- ☐ **C.** 控制
- ☐ **D.** 记录

第 2 题 以下关于Python语言的描述，错误的是（ ）。

- ☐ **A.** Python提供了常用的数据结构，并支持面向对象编程
- ☐ **B.** Python是解释型语言
- ☐ **C.** Python是一种高级程序设计语言
- ☐ **D.** Python程序在运行前需要预先编译

第 3 题 下列Python代码执行后输出的是（ ）。

```
lstA = [1,2,3]
lstB = [4,5,6,lstA]
del lstA
print(lstB)
```

- ☐ **A.** `[4, 5, 6, lstA]`
- ☐ **B.** `[4, 5, 6, 1, 2, 3]`
- ☐ **C.** `[4, 5, 6, [1, 2, 3]]`
- ☐ **D.** 执行将报错，因为 `lstA` 已经被删除

第 4 题 有关下面Python代码说法错误的是 ( )。

```
#sumA()和sumB()用于求1~N之和
def sumA(N):
    ret = 0
    for i in range(1, N + 1):
        ret += i
    return ret

def sumB(N):
    if N == 1:
        return 1
    else:
        return N + sumB(N - 1)

N = int(input("请输入大于等于1的正整
数:"))
print(sumA(N),sumB(N))
```

- [ ] **A.** `sumA()` 用循环方式求1 ~ N之和， `sumB()` 用递归方式求1 ~ N之和
- [ ] **B.** 默认情况下，倒数第二行被执行时如果输入较小的正整数如 `100` ，即倒数第一行能正确被执行，则能实现求1到N之和
- [ ] **C.** 默认情况下，倒数第二行被执行时如果输入较大的正整数如 `10000` ，倒数第一行被能正确被执行，能实现求1到N之和
- [ ] **D.** 默认情况下，一般说来， `sumA()` 和效率高于 `sumB()`

第 5 题 下面Python代码以递归方式实现字符串反序，横线处应填上代码是 ( )。

```
#字符串反序
def sReverse(sIn):
    if len(sIn) <= 1:
        return sIn
    else:
        return _____________________

print(sReverse("Hello"))
```

- [ ] **A.** `sReverse(sIn[1:]) + sIn[:1]`
- [ ] **B.** `sReverse(sIn[:1]) + sIn[1:]`
- [ ] **C.** `sIn[:1] + sReverse(sIn[1:])`
- [ ] **D.** `sIn[1:] + sReverse(sIn[:1])`

第 6 题 印度古老传说：创世时有三根金刚柱，其中一柱从下往上按照大小顺序摞着64片黄金圆盘，当圆盘逐一从一柱借助另外一柱全部移动到另外一柱时，宇宙毁灭。移动规则：在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。下面的Python代码以递归方式实现汉诺塔，横线处应填入代码是( )。

```
#递归实现汉诺塔，将N个圆盘从A通过B移动C
#圆盘从底到顶，半径必须从大到小
def Hanoi(A, B, C, N):
    if N == 1:
        print(A, "->", C)
    else:
        Hanoi(A, C, B, N-1)
        print(A, "->", C)
        ____________________________

Hanoi("甲", "乙", "丙" , 3)
```

- A. `Hanoi(B, C, A, N-2)`
- B. `Hanoi(B, A, C, N-1)`
- C. `Hanoi(A, B, C, N-2)`
- D. `Hanoi(C, B, A, N-1)`

第 7 题 根据下面Python代码的注释，横线处应填入( )。

```
isOdd = lambda N: N % 2 == 1

lstA = list(range(1,100))
lstA.sort(key = ___________)#lstA成员
偶数在前，奇数在后
lstB = [x for x in lstA if _________]
#lstB成员全为奇数
print(lstA, lstB)
```

- A. `isOdd isOdd(x)`
- B. `isOdd(x) isOdd`
- C. `isOdd isOdd`
- D. `isOdd(x) isOdd(x)`

第 8 题 有关下面Python代码正确的是（ ）。

```
def isEven(N):
    return N % 2 == 0

def checkNum(Fx,N):
    return Fx(N)

print(checkNum(isEven,10))
```

- [ ] **A.** `checkNum()` 函数定义错误
- [ ] **B.** 最后一行代码将 `isEven` 作为 `checkNum()` 参数将导致错误
- [ ] **C.** 最后一行代码执行后将输出 `True`
- [ ] **D.** 触发异常

第 9 题 有关下面Python代码正确的是（ ）。

```
isOdd = lambda N: N % 2 == 1

def Add(N,M):
    return N+M

def checkNum(Fx):
    return Fx

print(checkNum(isOdd)(10))
print(checkNum(Add)(10,20))
```

- [ ] **A.** `checkNum()` 函数定义错误
- [ ] **B.** 倒数第2行代码将 `isOdd` 作为 `checkNum()` 参数将导致错误
- [ ] **C.** 最后一行代码将 `Add` 作为 `checkNum()` 参数将导致错误
- [ ] **D.** 倒数两行代码执行后都将没有错误

第 10 题 下面代码执行后的输出是（ ）。

```
def jumpFloor(N):
    print(N, end = "#")
    if N == 1 or N == 2:
        return N
    else:
        return jumpFloor(N-1) +
jumpFloor(N-2)

print(jumpFloor(4))
```

- [ ] **A.** `4#3#2#2#4`
- [ ] **B.** `4#3#2#2#1#5`
- [ ] **C.** `4#3#2#1#2#4`
- [ ] **D.** `4#3#2#1#2#5`

第 11 题 下面Python代码中的 `isPrimeA()` 和 `isPrimeB()` 都用于判断参数N是否为素数，有关其时间复杂度的正确说法是（ ）。

```
def isPrimeA(N):
    if N < 2:
        return False

    for i in range(2,N):
        if N % i == 0:
            return False
    return True

def isPrimeB(N):
    if N < 2:
        return False

    endNum = int(N ** 0.5)
    for i in range(2,endNum+1):
        if N % i == 0:
            return False
    return True

print(isPrimeA(13),isPrimeB(13))
```

- ☐ **A.** `isPrimeA()` 的最坏时间复杂度是$O(N)$，`isPrimeB()` 的最坏时间复杂度是$O(\log N)$，`isPrimeA()` 优于 `isPrimeB()`
- ☐ **B.** `isPrimeA()` 的最坏时间复杂度是$O(N)$，`isPrimeB()` 的最坏时间复杂度是$O(N^{\frac{1}{2}})$，`isPrimeB()` 优于 `isPrimeA()`
- ☐ **C.** `isPrimeA()` 的最坏时间复杂度是$O(N^{\frac{1}{2}})$，`isPrimeB()` 的最坏时间复杂度是$O(N)$，`isPrimeA()` 优于 `isPrimeB()`
- ☐ **D.** `isPrimeA()` 的最坏时间复杂度是$O(\log N)$，`isPrimeB()` 的最坏时间复杂度是$O(N)$，`isPrimeB()` 优于 `isPrimeB()`

第 12 题 下面Python代码用于归并排序，其中 `merge()` 函数被调用次数为（ ）。

```
def mergeSort(listData):
    if len(listData) <= 1:
        return listData

    Middle = len(listData) // 2
    Left, Right = mergeSort(listData[:Middle]),
mergeSort(listData[Middle:])

    return merge(Left, Right)
```

```
def merge(Left, Right):
    Result = []
    i, j = 0 , 0

    while i < len(Left) and j < len(Right):
        if Left[i] <= Right[j]:
            Result.append(Left[i])
            i += 1
        else:
            Result.append(Right[j])
            j += 1

    Result.extend(Left[i:])
    Result.extend(Right[j:])

    return Result

lstA = [1, 3, 2, 7, 11, 5, 3]
lstA = mergeSort(lstA)

print(lstA)
```

- ☐ **A.** 0
- ☐ **B.** 1
- ☐ **C.** 6
- ☐ **D.** 7

第 13 题 在上题的归并排序算法中，代码 `Left, Right = mergeSort(listData[:Middle]), mergeSort(listData[Middle:])` 涉及到的算法有（ ）。

- ☐ **A.** 搜索算法
- ☐ **B.** 分治算法
- ☐ **C.** 贪心算法
- ☐ **D.** 递推算法

第 14 题 归并排序算法的基本思想是（ ）。

- ☐ **A.** 将数组分成两个子数组，分别排序后再合并。
- ☐ **B.** 随机选择一个元素作为枢轴，将数组划分为两个部分。
- ☐ **C.** 从数组的最后一个元素开始，依次与前一个元素比较并交换位置。
- ☐ **D.** 比较相邻的两个元素，如果顺序错误就交换位置。

第 15 题 有关下面Python代码的说法正确的是（ ）。

```
class Node:
    def __init__(self, Val, Nxt = None):
        self.Value = Val
        self.Next = Nxt

firstNode = Node(10)
firstNode.Next = Node(100)
firstNode.Next.Next = Node(111,firstNode)
```

- [ ] **A.** 上述代码构成单向链表
- [ ] **B.** 上述代码构成双向链表
- [ ] **C.** 上述代码构成循环链表
- [ ] **D.** 上述代码构成指针链表

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 TCP/IP的传输层的两个不同的协议分别是UDP和TCP。

第 2 题 在特殊情况下流程图中可以出现三角框和圆形框。

第 3 题 找出自然数N以内的所有质数常用埃氏筛法，其时间复杂度为$O(N)$。

第 4 题 Python的in运算符相当于通过查找算法判断元素是否存在，其查找通常采用二分法。

第 5 题 在以下Python代码中，最后一行代码执行时将报错，因为y所代表的数已经被删除。

```
x = [1, 2, 3, [4, 5, 6]]
y = x[3]

del x[3]
print(y)
```

第 6 题 贪心算法的解可能不是最优解。

第 7 题 一般说来，冒泡排序算法优于归并排序。

第 8 题 Python的内置函数 `sorted()` 可以对支持 `for-in` 循环的 `str`、`list`、`tuple`、`dict`、`set` 排序，且是稳定排序。

第 9 题 下面的Python代码将输出0-99（包含0和99）之间的整数，顺序随机。

```
import random
print(sorted(range(100), key = lambda x:
random.random()))
```

第 10 题

下面的Python代码执行后将输出 `[0, 5, 1, 6, 2, 3, 4]` 。

```
lst = list(range(7))
sorted(lst, key = lambda x: x%5)
print(lst)
```

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题编号**：2023-09-23-05-P-01
- **试题名称**：因数分解
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.1.1　问题描述

每个正整数都可以分解成素数的乘积，例如：$6 = 2 \times 3$、$20 = 2^2 \times 5$

现在，给定一个正整数N，请按要求输出它的因数分解式。

3.1.2　输入描述

输入第一行，包含一个正整数N。约定$2 \leq N \leq 10^{12}$

3.1.3　输出描述

输出一行，为N的因数分解式。要求按质因数由小到大排列，乘号用星号*表示，且左右各空一格。当且仅当一个素数出现多次时，将它们合并为指数形式，用上箭头^表示，且左右不空格。

3.1.4　样例输入1

```
6
```

3.1.5　样例输出1

```
2 * 3
```

3.1.6 样例输入2

```
20
```

3.1.7 样例输出2

```
2^2 * 5
```

3.1.8 样例输入3

```
23
```

3.1.9 样例输出3

```
23
```

3.1.10 参考程序

```
input_number = int(input())
original_number = input_number
factors = []
def isPrime(num):
    for j in range(2, int(num**0.5)+1):
        if num % j == 0:
            return False
    return True
if isPrime(input_number):
    factors.append([input_number, 1])
else:
    for i in range(2, input_number+1):
        input_number = int(input_number)
        if input_number % i == 0:
            factors.append([i, 0])
            while input_number % i == 0:
                factors[-1][1] += 1
                input_number //= i
            if isPrime(input_number):
                if input_number != 1:
                    factors.append([input_number, 1])
                break

for i in range(len(factors)):
    if i != 0:
        print(" * ", end="")
    if factors[i][1] == 1:
        print(factors[i][0], end="")
    else:
        print(f"{factors[i][0]}^{factors[i][1]}", end="")
print("")
```

3.2 编程题 2

- **试题编号**：2023-09-23-05-C-02
- **试题名称**：巧夺大奖
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.2.1 问题描述

小明参加了一个巧夺大奖的游戏节目。主持人宣布了游戏规则：

1、游戏分为n个时间段，参加者每个时间段可以选择一个小游戏。

2、游戏中共有n个小游戏可供选择。

3、每个小游戏有规定的时限和奖励。对于第i个小游戏，参加者必须在第T_i个时间段结束前完成才能得到奖励R_i。

小明发现，这些小游戏都很简单，不管选择哪个小游戏，他都能在一个时间段内完成。关键问题在于，如何安排每个时间段分别选择哪个小游戏，才能使得总奖励最高？

3.2.2 输入描述

输入第一行，包含一个正整数n。n既是游戏时间段的个数，也是小游戏的个数。约定$1 \leq n \leq 500$。

输入第二行，包含n个正整数。第i个正整数为T_i，即第i个小游戏的完成期限。约定$1 \leq T_i \leq n$。

输入第三行，包含n个正整数。第i个正整数为R_i，即第i个小游戏的完成奖励。约定$1 \leq R_i \leq 1000$。

3.2.3 输出描述

输出一行，包含一个正整数C，为最高可获得的奖励。

3.2.4 样例输入1

```
7
4 2 4 3 1 4 6
70 60 50 40 30 20 10
```

3.2.5 样例输出1

```
230
```

3.2.6 样例解释1

7个时间段可分别安排完成第4、2、3、1、6、7、5个小游戏，其中第4、2、3、1、7个小游戏在期限内完成。因此，可以获得总计$40 + 60 + 50 + 70 + 10 = 230$的奖励。

3.2.7 参考程序

```python
total_segements = int(input())
limits = []
rewards = []
s_limits = input()
s_limits = s_limits.split(' ')
s_rewards = input()
```

```
s_rewards = s_rewards.split(" ")
for item in s_limits:
    limits.append(int(item))
for item in s_rewards:
    rewards.append(int(item))
games = []
for i in range(len(s_limits)):
    # 结束时间，奖励值，是否可以选，是否选过
    games.append([limits[i], rewards[i], False, False])

def check_chooseable(states, current):
    for item in states:
        if item[0] >= current:
            item[2] = True

def get_current_max(states, current):
    maximum = 0
    chosen = None
    for i in range(len(states)):
        if states[i][2] and not states[i][3]:
            if maximum < states[i][1]:
                #print(states[i][1])
                maximum = max(states[i][1], maximum)
                chosen = i
    if chosen is not None:
        states[chosen][3] = True
    return maximum

total = 0
#print(games)
for t in range(total_segements, 0, -1):
    #print(t)
    check_chooseable(games, t)
    total += get_current_max(games, t)
    #print(games)
print(total)
```