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

Python　六级

2024 年 09 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	A	C	A	B	B	B	A	C	C	A	A	B	B	C

第 1 题 以下（ ）没有涉及Python语言的面向对象特性支持。

- **A.** Python中构造一个 `class`
- **B.** Python中调用 `printf` 函数
- **C.** Python中调用用户定义的类成员函数
- **D.** Python中构造来源于同一基类的多个派生类

第 2 题 关于Python中面向对象的类的继承，下面说法错误的是（ ）

- **A.** 子类可以通过继承不能访问到父类的所有属性
- **B.** 多个子类可以继承同一个父类
- **C.** 子类和子类产生的对象都可以通过句点的方式拿到父类
- **D.** python中支持多继承

第 3 题 有6个元素，按照 6,5,4,3,2,1 的顺序进入栈S，下列（ ）的出栈序列是不能出现的（ ）。

- **A.** 5,4,3,6,1,2
- **B.** 4,5,3,1,2,6
- **C.** 3,4,6,5,2,1
- **D.** 2,3,4,1,5,6

第 4 题 采用如下代码实现检查输入的字符串括号是否匹配，横线上应填入的代码为（ ）。

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return not self.items
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
```

```
        def peek(self):
        if not self.is_empty():
            return self.items[-1]
     def size(self):
        return len(self.items)
 def paren_match(expr):
    s = Stack()
    balanced = True
    index = 0
    while index < len(expr) and balanced:
        symbol = expr[index]
        if symbol in '([{':
            _____________
        else:
            if s.is_empty():
                balanced = False
            else:
                top = s.pop()
                if not matches(top, symbol):
                    balanced = False
        index += 1

    if balanced and s.is_empty():
        return True
    else:
        return False
def matches(opening, closing):
    opens = '([{'
    closers = ')]}'
    return opening in opens and closers.index(closing) == opens.index(opening)
```

- [ ] **A.** `s.push(symbol)`
- [ ] **B.** `s.pop(symbol)`
- [ ] **C.** `s.push(index)`
- [ ] **D.** `s.pop(index)`

第 5 题 下面代码判断队列的第一个元素是否等于a，并删除该元素，横向上应填写（ ）。

```
import queue
q = queue.Queue()
a = 'a'
if ____________________________________
    q.get()
    print('元素 {} 是队列的第一个元素，并已被移除。'.format(a))
else:
    print('队列的第一个元素不是 {}.'.format(a))
```

- [ ] **A.** `not q.empty() and q.queue[0] != a:`
- [ ] **B.** `not q.empty() and q.queue[0] == a:`
- [ ] **C.** `q.empty() and q.queue[0] == a:`

- [ ] **D.** `q.empty() and q.queue[0] != a:`

第 6 题 假设字母表 `{a,b,c,d,e}` 在字符串出现的频率分别为 10%，15%，30%，16%，29%。若使用哈夫曼编码方式对字母进行二进制编码，则字符 `abcdef` 分别对应的一组哈夫曼编码的长度分别为（ ）。

- [ ] **A.** 4, 4, 1, 3, 2
- [ ] **B.** 3, 3, 2, 2, 2
- [ ] **C.** 3, 3, 1, 2, 1
- [ ] **D.** 4, 4, 1, 2, 2

第 7 题 以下Python代码实现n位的格雷码，则横线上应填写（ ）。

```
def generate_gray_code(n):
    if n <= 0:
        return []
    if n == 1:
        return [0, 1]

    gray_code = generate_gray_code(n - 1)
    ____________________________________
    for x in gray_code]
    return gray_code + inverted_gray_code
```

- [ ] **A.** `inverted_gray_code = [int(('0' * n + bin(x)[2:])[-n:], 2)`
- [ ] **B.** `inverted_gray_code = [int(('1' * n + bin(x)[2:])[-n:], 2)`
- [ ] **C.** `inverted_gray_code = [int(('1' * n + bin(x)[1:])[-n:], 2)`
- [ ] **D.** `inverted_gray_code = [int(('1' * n + bin(x)[2:])[n:], 2)`

第 8 题 给定一棵二叉树，其前序遍历结果为：ABDECFG,中序遍历结果为：DEBACFG，则这棵树的正确后序遍历结果是（ ）。

- [ ] **A.** EDBGFCA
- [ ] **B.** EDGBFCA
- [ ] **C.** DEBGFCA
- [ ] **D.** DBEGFCA

第 9 题 一棵有n个结点的完全二叉树用数组进行存储与表示，已知根结点存储在数组的第1个位置。若存储在数组第9个位置的结点存在兄弟结点和两个子结点，则它的兄弟结点和右子结点的位置分别是（ ）。

- [ ] **A.** 8, 18
- [ ] **B.** 10, 18
- [ ] **C.** 8, 19
- [ ] **D.** 10, 19

第 10 题 二叉树的深度定义为从根结点到叶结点的最长路径上的结点数，则以下基于二叉树的深度优先搜索实现的深度计算函数中横线上应填写（ ）。

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def max_depth(root_node):
    if root_node is None:
        return 0
    else:
        left_depth = max_depth(root_node.left)
        right_depth = max_depth(root_node.right)
        ____________________________________
```

- [ ] **A.** `return max(left_depth, right_depth)`
- [ ] **B.** `return min(left_depth, right_depth) + 1`
- [ ] **C.** `return max(left_depth, right_depth) + 1`
- [ ] **D.** `return max(left_depth, right_depth) - 1`

第 11 题 以下基于二叉树的搜索实现的深度计算函数中横线上应填写（ ）。

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def height(root):
    if root is None:
        return 0
    else:
        left_height = height(root.left)
        right_height = height(root.right)
        ____________________________________
```

- [ ] **A.** `return max(left_height, right_height) + 1`
- [ ] **B.** `return min(left_height, right_height) - 1`
- [ ] **C.** `return min(left_height, right_height) + 1`
- [ ] **D.** `return max(left_height, right_height) - 1`

第 12 题 二叉搜索树中的每个结点，其左子树的所有结点值都小于该结点值，右子树的所有结点值都大于该结点值。以下代码对给定的整数数组(假设数组中没有数值相等的元素)，构造一个对应的二叉搜索树，横线上应填写（ ）：

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

```
class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        def helper(node, min_val, max_val):
            if not node:
                return True
            ____________________________________________
                return False
            return helper(node.left, min_val, node.val) and helper(node.right,
node.val, max_val)

        return helper(root, float('-inf'), float('inf'))
```

- [ ] **A.** `if node.val <= min_val or node.val >= max_val:`
- [ ] **B.** `if node.val >= min_val or node.val >= max_val:`
- [ ] **C.** `if node.val <= min_val or node.val <= max_val:`
- [ ] **D.** `if node.val >= min_val or node.val <= max_val:`

第 13 题 对上题中的二叉搜素树，当输入数组为$[5,3,7,2,4,6,8]$时，构建二叉搜索树，并采用如下代码实现的遍历方式，得到的输出是（ ）。

```
def traversal(tree_node* root) :
    if (root == nullptr) {
        return
    }

traversal(root->left)
print(root->val)
print(" ")
traversal(root->right)
```

- [ ] **A.** 5 3 7 2 4 6 8
- [ ] **B.** 2 3 4 5 6 7 8
- [ ] **C.** 2 4 3 6 8 7 5
- [ ] **D.** 2 4 3 5 6 7 8

第 14 题 动态规划通常用于解决（ ）。

- [ ] **A.** 无法分解的问题
- [ ] **B.** 可以分解成相互依赖的子问题的问题
- [ ] **C.** 可以通过贪心算法解决的问题
- [ ] **D.** 只能通过递归解决的问题

第 15 题 阅读以下用动态规划解决的0-1背包问题的函数，假设背包的容量W是10kg，假设输入4个物品的重量$weights$分别为$1,3,4,6$（单位为kg），每个物品对应的价值$values$分别为$20,30,50,60$，则函数的输出为（ ）。

```
def knapsack(capacity, weights, values):
    dp = [[0 for _ in range(capacity + 1)] for _ in range(len(weights) + 1)]
```

```
    for i in range(1, len(weights) + 1):
        for j in range(1, capacity + 1):
            if weights[i - 1] <= j:
                    dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] +
values[i - 1])
            else:
                dp[i][j] = dp[i - 1][j]
    return dp[-1][-1]

weights = [1, 3, 4,6]
values = [20,30,50,60]
capacity = 10
print(knapsack(capacity, weights, values))
```

- [ ] **A.** 90
- [ ] **B.** 100
- [ ] **C.** 110
- [ ] **D.** 140

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 C++、Python和JAVA等都是面向对象的编程语言。

第 2 题 在python中，类的静态成员变量只能被该类对象的成员函数访问。

第 3 题 栈和队列均可通过数组或链表来实现，其中数组实现支持随机访问、占用内存较少，但插入和删除元素效率低；链表实现的元素插入与删除效率高，但元素访问效率低、占用内存较多。

第 4 题 运行以下python代码，屏幕将输出“derived class”。

```
class BaseClass:
    def my_method(self):
        print("base class")

class DerivedClass(BaseClass):
    def my_method(self):
        print("derived class")

derived_instance = DerivedClass()
derived_instance.my_method()
```

第 5 题 如下列代码所示的基类（base）及其派生类（derived），则生成一个派生类的对象时，只调用派生类的构造函数。

```python
class Base:
    def __init__(self):
        print("Base.__init__ called")

class Derived(Base):
    def __init__(self):
        print("Derived.__init__ called")
        super().__init__()
d = Derived()
```

第 6 题 哈夫曼编码本质上是一种贪心策略。

第 7 题 如果根结点的深度记为1，则一棵恰有2024个叶结点的二叉树的深度最少是12。

第 8 题 在非递归实现的树的广度优先搜索中，通常使用栈来辅助实现。

第 9 题 状态转移方程是动态规划的核心，可以通过递推方式表示问题状态的变化。

第 10 题 应用动态规划算法时，识别并存储重叠子问题的解是必须的。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：小杨和整数拆分
- **时间限制**：20.0 s
- **内存限制**：512.0 MB

3.1.1 题面描述

小杨有一个正整数 n，小杨想将它拆分成若干完全平方数的和，同时小杨希望拆分的数量越少越好。

小杨请你编写程序计算出总和为 n 的完全平方数的最少数量。

3.1.2 输入格式

第一行包含一个正整数 n，含义如题面所示。

3.1.3 输出格式

输出一个整数，代表总和为 n 的完全平方数的最少数量。

3.1.4 样例1

```
18
```

```
2
```

$18 = 9 + 9 = 16 + 1 + 1$，其中最少需要 2 个完全平方数。

子任务编号	数据点占比	n
1	20%	≤ 20
2	40%	≤ 1000
3	40%	$\leq 10^5$

对于全部数据，保证有 $1 \leq n \leq 10^5$。

3.1.5 参考程序

```python
import math

N = int(1e5) + 10
dp = [0] * N

n = int(input())

for i in range(1, n + 1):
    dp[i] = i
    for j in range(1, int(math.sqrt(i)) + 1):
        dp[i] = min(dp[i - j * j] + 1, dp[i])

print(dp[n])
```

3.2 编程题 2

- **试题名称**：算法学习
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题面描述

小杨计划学习 m 种算法，为此他找了 n 道题目来帮助自己学习，每道题目至多学习一次。

小杨对于 m 种算法的初始掌握程度均为 0。第 i 道题目有对应的知识点 a_i，即学习第 i 道题目可以令小杨对第 a_i 种算法的掌握程度提高 b_i。小杨的学习目标是对 m 种算法的掌握程度均至少为 k。

小杨认为连续学习两道相同知识点的题目是不好的，小杨想请你编写程序帮他计算出他最少需要学习多少道题目才能使得他在完成学习目标的同时避免连续学习两道相同知识点的题目。

3.2.2 输入格式

第一行三个正整数 m, n, k，代表算法种类数，题目数和目标掌握程度。

第二行 n 个正整数 $a_1, a_2, a_3, \ldots, a_n$，代表每道题目的知识点。

第二行 n 个正整数 $b_1, b_2, b_3, \ldots, b_n$，代表每道题目提升的掌握程度。

3.2.3 输出格式

输出一个整数，代表小杨最少需要学习题目的数量，如果不存在满足条件的方案，输出 -1。

3.2.4 样例1

```
3 5 10
1 1 2 3 3
9 1 10 10 1
```

```
4
```

3.2.5 样例2

```
2 4 10
1 1 1 2
1 2 7 10
```

```
-1
```

对于样例1，一种最优学习顺序为第一道题，第三道题，第四道题，第二道题。

子任务编号	数据点占比	m	n	b_i	k
1	30%	$=2$	≤ 9	≤ 10	≤ 10
2	30%	≤ 9	≤ 9	≤ 10	≤ 10
3	40%	$\leq 10^5$	$\leq 10^5$	$\leq 10^5$	$\leq 10^5$

对于全部数据，保证有 $1 \leq m,n \leq 10^5, 1 \leq b_i, k \leq 10^5, 1 \leq a_i \leq m$。

3.2.6 参考程序

```python
N = int(1e5) + 5

def cmp(i, j):
    return i > j

m,n, k = map(int, input().split())
score = [[] for _ in range(m + 2)]
a = [0] * n
b = [0] * n

s = input().split()
for i in range(n):
    a[i] = int(s[i])

s = input().split()
for i in range(n):
    b[i] = int(s[i])
    score[a[i]].append(b[i])

need = [0] * (m + 2)
ans = 0
mx_meed = 0
mx_need_i = -1

for i in range(1, m + 1):
    score[i].sort(reverse=True)
    sum_vals = 0
    for j in range(len(score[i])):
        sum_vals += score[i][j]
        if sum_vals >= k:
            need[i] = j + 1
            break

    if sum_vals < k:
        print("-1")
        exit(0)
```

```
37
38      ans += need[i]
39      if need[i] > mx_meed:
40          mx_meed = need[i]
41          mx_need_i = i
42
43  if mx_meed - 1 <= ans - mx_meed:
44      print(ans)
45  else:
46      last = 0
47      for i in range(1, m + 1):
48          if i != mx_need_i:
49              last += len(score[i]) - need[i]
50
51      print(2 * mx_meed - 1 if mx_meed - 1 <= ans - mx_meed + last else -1)
```