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

Python 六级

2025 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	C	B	C	B	A	B	A	A	B	B	D	A	A	C	D

第 1 题 关于Python类的说法，错误的是（ ）。

- **A.** 构造方法（ `__init__` ）不能声明为虚方法，但析构方法（ `__del__` ）可以。
- **B.** 函数参数传传递的是对象的引用，不会复制对象。
- **C.** 静态方法属于类、不属于对象，因此不能使用 `对象.方法(...)` 的形式调用静态方法。
- **D.** 当派生类对象被销毁时，不会自动调用基类的 `__del__()` 。

第 2 题 下面代码执行结果是（ ）。

```python
class Vehicle:
    def __init__(self, brand):
        self._brand = brand  # 私有属性用下划线表示

    def set_brand(self, brand):
        self._brand = brand

    def get_brand(self):
        return self._brand

    def move(self):
        print(f"{self._brand} is moving...")

class Car(Vehicle):
    def __init__(self, brand, seat_count):
        super().__init__(brand)
        self._seat_count = seat_count

    def show_info(self):
        print(f"This car is a {self.get_brand()} with {self._seat_count} seats.")

    def move(self):
        print(f"{self.get_brand()} car is driving on the road!")

def main():
    v1 = Car("Toyota", 5)

    v1.move()

if __name__ == "__main__":
    main()

```

- **A.** `Toyota is moving...`

- [ ] **B.** `Toyota car is driving on the road!`
- [ ] **C.** 编译错误
- [ ] **D.** 运行结果不确定

第 3 题 下面代码中 `v1` 和 `v2` 调用了相同接口 `move()`，但输出结果不同，这体现了面向对象编程的（ ）特性。

```python
class Vehicle:
    def __init__(self, brand):
        self._brand = brand  # 私有属性用下划线表示

    def set_brand(self, brand):
        self._brand = brand

    def get_brand(self):
        return self._brand

    def move(self):
        print(f"{self._brand} is moving...")

class Car(Vehicle):
    def __init__(self, brand, seat_count):
        super().__init__(brand)
        self._seat_count = seat_count

    def show_info(self):
        print(f"This car is a {self.get_brand()} with {self._seat_count} seats.")

    def move(self):
        print(f"{self.get_brand()} car is driving on the road!")

class Bike(Vehicle):
    def __init__(self, brand):
        super().__init__(brand)

    def move(self):
        print(f"{self.get_brand()} bike is cycling on the path!")

def main():
    v1 = Car("Toyota", 5)
    v2 = Bike("Giant")

    v1.move()  # 输出: Toyota car is driving on the road!
    v2.move()  # 输出: Giant bike is cycling on the path!

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

- [ ] **A.** 继承 (Inheritance)
- [ ] **B.** 封装 (Encapsulation)
- [ ] **C.** 多态 (Polymorphism)
- [ ] **D.** 链接 (Linking)

第 4 题 栈的操作特点是（ ）。

- [ ] **A.** 先进先出
- [ ] **B.** 先进后出
- [ ] **C.** 随机访问
- [ ] **D.** 双端进出

第 5 题 循环队列常用于实现数据缓冲。假设一个循环队列容量为 `5`（即最多存放 `4` 个元素，留一个位置区分空与满），依次进行操作：入队数据1，2，3，出队1个数据，再入队数据4和5，此时队首到队尾的元素顺序是（ ）。

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

第 6 题 以下函数 `createTree()` 构造的树是什么类型?

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

def create_tree():
    root = TreeNode(1)
    root.left = TreeNode(2)
    root.right = TreeNode(3)
    root.left.left = TreeNode(4)
    root.left.right = TreeNode(5)
    return root
```

- [ ] **A.** 满二叉树
- [ ] **B.** 完全二叉树
- [ ] **C.** 二叉排序树
- [ ] **D.** 其他都不对

第 7 题 已知二叉树的 中序遍历 是 [D, B, E, A, F, C]，先序遍历 是 [A, B, D, E, C, F]。请问该二叉树的后序遍历结果是（ ）。

- [ ] **A.** [D, E, B, F, C, A]
- [ ] **B.** [D, B, E, F, C, A]
- [ ] **C.** [D, E, B, C, F, A]
- [ ] **D.** [B, D, E, F, C, A]

第 8 题 完全二叉树可以用数组连续高效存储。如果节点从 `1` 开始编号，则对有两个孩子节点的节点 `i`，（ ）。

- [ ] **A.** 左孩子位于 `2i`，右孩子位于 `2i+1`
- [ ] **B.** 完全二叉树的叶子节点可以出现在最后一层的任意位置
- [ ] **C.** 所有节点都有两个孩子
- [ ] **D.** 左孩子位于 `2i+1`，右孩子位于 `2i+2`

第 9 题 设有字符集 `{a, b, c, d, e, f}`，其出现频率分别为 `{5, 9, 12, 13, 16, 45}`。哈夫曼算法构造最优前缀编码，以下哪一组可能是对应的哈夫曼编码?（非叶子节点左边分支记作 0，右边分支记作 1，左右互换不影响正确性）。

- [ ] **A.** a: 00；b: 01；c: 10；d: 110；e: 111；f: 0
- [ ] **B.** a: 1100；b: 1101；c: 100；d: 101；e: 111；f: 0
- [ ] **C.** a: 000；b: 001；c: 01；d: 10；e: 110；f: 111
- [ ] **D.** a: 10；b: 01；c: 100；d: 101；e: 111；f: 0

第 10 题 下面代码生成格雷编码中，`gray_code` 的目的是生成所有的长度为n位的格雷码,则横线上应填写（ ）。

```
def gray_code(n):
    if n == 0:
        return ["0"]
    if n == 1:
        return ["0", "1"]

    prev = gray_code(n-1)
    result = []
    for s in prev:
        ____________________________
    for i in range(len(prev)-1, -1, -1):
        result.append("1" + prev[i])
    return result
```

- A. `result.append("1" + s)`
- B. `result.append("0" + s)`
- C. `result.append("s" + 0)`
- D. `result.append("0" + 0)`

第 11 题 请将下列树的深度优先遍历代码补充完整，横线处应填入（ ）。

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

def dfs(root):
    if not root:
        return

    ________________________
    stack.append(root)

    while stack:
        node = stack.pop()
        print(node.val, end=" ")

        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)
```

- A. `vector=[]`
- B. `list=[]`
- C. `queue=[]`
- D. `stack = []`

第 12 题 令 n 是树的节点数目，下列代码实现了树的广度优先遍历，其时间复杂度是（ ）。

```
from collections import deque

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

```
def bfs(root):
    if not root:
        return

    q = deque()
    q.append(root)

    while q:
        node = q.popleft()
        print(node.val, end=" ")

        if node.left:
            q.append(node.left)
        if node.right:
            q.append(node.right)
```

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

第 13 题 在二叉搜索树中查找元素 `50`，从根结点开始：若根值为 `60`，则下一步应去：

- [ ] **A.** 左子树
- [ ] **B.** 右子树
- [ ] **C.** 随机
- [ ] **D.** 根结点

第 14 题 删除二叉排序树节点时，如果节点有两个孩子，则横线处应填入（ ），其中 `findMax` 和 `findMin` 分别为找树的最大值和最小值。

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

def find_min(node):
    """在二叉搜索树中找到最小节点"""
    current = node
    while current and current.left:
        current = current.left
    return current

def delete_node(root, key):
    if not root:
        return None

    if key < root.val:
        root.left = delete_node(root.left, key)
    elif key > root.val:
        root.right = delete_node(root.right, key)
    else:
        if not root.left:
            return root.right
        elif not root.right:
            return root.left
        else:
            temp = find_min(____________)
```

```
            root.val = temp.val
            root.right = delete_node(root.right, temp.val)
```

- [ ] **A.** `root.left`
- [ ] **B.** `root.left.right`
- [ ] **C.** `root.right`
- [ ] **D.** `root.right.left`

第 15 题 给定n个物品和一个最大承重为W的背包，每个物品有一个重量$wt[i]$和价值$val[i]$，每个物品只能选择放或不放。目标是选择若干个物品放入背包，使得总价值最大，且总重量不超过W，则横线上应填写（ ）。

```
def knapsack(W, wt, val, n):
    dp = [0] * (W + 1)
    for i in range(n):
        for w in range(W, wt[i] - 1, -1):
            ____________________________
    return dp[W]
```

- [ ] **A.** `dp[w] = max(dp[w], dp[wt[i]] + val[i])`
- [ ] **B.** `dp[w] = max(dp[w], dp[w - wt[0]] + val[i])`
- [ ] **C.** `dp[w] = max(dp[w], dp[w + wt[i]] - val[i])`
- [ ] **D.** `dp[w] = max(dp[w], dp[w - wt[i]] + val[i])`

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 在 Python 中，类的方法默认是“虚函数”，派生类只要重写方法。如果想复用基类逻辑时，可显式调用基类对应的函数。

第 2 题 哈夫曼编码是最优前缀码，且编码结果唯一。

第 3 题 一个含有 100 个结点的完全二叉树，高度为 8。

第 4 题 栈的 `pop` 操作返回栈顶元素并移除它。

第 5 题 循环队列通过模运算循环使用空间。

第 6 题 一棵有 n 个结点的二叉树一定有 $n-1$ 条边。

第 7 题 以下代码实现了二叉树的中序遍历，输入以下二叉树，中序遍历结果是 `4 2 5 1 3 6` 。

```
#     1
#    / \
#   2   3
#  / \   \
# 4   5   6
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def inorder_iterative(root):
    stack = []
    curr = root
```

```
    while curr or stack:
        while curr:
            stack.append(curr)
            curr = curr.left

        curr = stack.pop()
        print(curr.val, end=" ")

        curr = curr.right
```

第 8 题 下面代码实现的二叉搜索树的查找操作时间复杂度是 $O(h)$，h为树高。

```
def searchBST(root, val):
    while root and root.val != val:
        root = root.left if val < root.val else root.right
    return root

```

第 9 题 下面代码实现了动态规划版本的斐波那契数列计算，其时间复杂度是 $O(2^n)$。

```
def fib_dp(n):
    if n <= 1:
        return n
    dp = [0] * (n + 1)
    dp[0] = 0
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
```

第 10 题 有一排香蕉，每个香蕉有不同的甜度值。小猴子想吃香蕉，但不能吃相邻的香蕉。以下代码能找到小猴子吃到最甜的香蕉组合。

```
def find_selected_bananas(bananas, dp):
    selected = []
    i = len(bananas) - 1

    while i >= 0:
        if i == 0:
            selected.append(0)
            break

        if dp[i] == dp[i-1]:
            i -= 1
        else:
            selected.append(i)
            i -= 2

    selected.reverse()
    print("小猴子吃了第: ", end="")
    for idx in selected:
        print(idx + 1, end=" ")
    print("个香蕉")

def main():
    bananas = [1, 2, 3, 1]  # 每个香蕉的甜度

    if not bananas:
        return

    n = len(bananas)
    dp = [0] * n
```

```
    dp[0] = bananas[0]

    if n > 1:
        dp[1] = max(bananas[0], bananas[1])

    for i in range(2, n):
        dp[i] = max(bananas[i] + dp[i-2], dp[i-1])

    find_selected_bananas(bananas, dp)

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

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：划分字符串
- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

小 A 有一个由 n 个小写字母组成的字符串 s。他希望将 s 划分为若干个子串，使得子串中每个字母至多出现一次。例如，对于字符串 `street` 来说，`str` + `e` + `e` + `t` 是满足条件的划分；而 `s` + `tree` + `t` 不是，因为子串 `tree` 中 `e` 出现了两次。

额外地，小 A 还给出了价值 $a_1, a_2, \ldots, a_n$，表示划分后长度为 i 的子串价值为 a_i。小 A 希望最大化划分后得到的子串价值之和。你能帮他求出划分后子串价值之和的最大值吗?

3.1.2 输入格式

第一行，一个正整数 n，表示字符串的长度。

第二行，一个包含 n 个小写字母的字符串 s。

第三行，n 个正整数 $a_1, a_2, \ldots, a_n$，表示不同长度的子串价值。

3.1.3 输出格式

一行，一个整数，表示划分后子串价值之和的最大值。

3.1.4 样例

3.1.4.1 输入样例 1

```
6
street
2 1 7 4 3 3
```

3.1.4.2 输出样例 1

```
13
```

3.1.4.3 输入样例 2

```
8
blossoms
1 1 2 3 5 8 13 21
```

3.1.4.4 输出样例 2

```
8
```

3.1.5 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 10^3$。

对于所有测试点，保证 $1 \leq n \leq 10^5$，$1 \leq a_i \leq 10^9$。

3.1.6 参考程序

```python
n = int(input())  # 读取字符串长度
s = input().strip()  # 读取原字符串
a = [0] + list(map(int, input().split()))  # 各个长度的字字符串的价值

# 本题的核心，f[i]表示字符串s前i个字符能够产生的最大分割收益
f = [0] * (n + 1)
for i in range(1, n + 1):
    # 计算字符串前i个字符的最大划分价值
    t = ''  # t表示最后一个子字符串包含的字符
    for j in range(i, 0, -1):
        # 遍历最后一个子字符串的长度
        cur = s[j - 1]  # cur 最后加入的字符的内容
        if cur in t:  # 如果目前即将加入的字符，已经在最后一个子字符串里了，就不再继续扩展最后一个子字符串了
            break
        t += cur
        # 核心步骤：前i个字符能产生的最大分割收益，要么是目前已经得到的最大价值，要么是最后一个划分长度为j的最大价值
        f[i] = max(f[i], f[j - 1] + a[i - j + 1])
print(f[n])
```

3.2 编程题 2

- **试题名称**：货物运输
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

A 国有 n 座城市，依次以 $1, 2, \ldots, n$ 编号，其中 1 号城市为首都。这 n 座城市由 $n-1$ 条双向道路连接，第 i 条道路（$1 \leq i < n$）连接编号为 u_i, v_i 的两座城市，道路长度为 l_i。任意两座城市间均可通过双向道路到达。

现在 A 国需要从首都向各个城市运送货物。具体来说，满载货物的车队会从首都开出，经过一座城市时将对应的货物送出，因此车队需要经过所有城市。A 国希望你设计一条路线，在从首都出发经过所有城市的前提下，最小化经过的道路长度总和。注意一座城市可以经过多次，车队最后可以不返回首都。

3.2.2 输入格式

第一行，一个正整数 n，表示 A 国的城市数量。

接下来 $n-1$ 行，每行三个整数 u_i, v_i, l_i，表示一条双向道路连接编号为 u_i, v_i 的两座城市，道路长度为 l_i。

3.2.3 输出格式

一行，一个整数，表示你设计的路线所经过的道路长度总和。

3.2.4 样例

3.2.4.1 输入样例 1

```
4
1 2 6
1 3 1
3 4 5
```

3.2.4.2 输出样例 1

```
18
```

3.2.4.3 输入样例 2

```
7
1 2 1
2 3 1
3 4 1
7 6 1
6 5 1
5 1 1
```

3.2.4.4 输出样例 2

```
9
```

3.2.5 数据范围

对于 30% 的测试点，保证 $1 \leq n \leq 8$。

对于另外 30% 的测试点，保证仅与一条双向道路连接的城市恰有两座。

对于所有测试点，保证 $1 \leq n \leq 10^5$，$1 \leq u_i, v_i \leq n$，$1 \leq l_i \leq 10^9$。

3.2.6 参考程序

```python
# 以下是参考代码1，因为要用到dfs和递归，而且路径可能会很长，所以需要设置最大递归长度
import sys
sys.setrecursionlimit(int(1.5e5))
n = int(input())  # 节点总数
e = [[] for _ in range(n + 1)]  # 邻接表，e[i]表示从i出发的边，每一条边的形式是[dst_node, length]
s = 0  # 全体边长总和

# 这个循环的目的是构建图的邻接表
for i in range(n - 1):
    u, v, w = map(int, input().split())
    e[u].append([v, w])
    e[v].append([u, w])
    s += w

# 寻找一个节点k这个节点到根节点（首都节点）的路径长度最长
```

```python
# u: 目前的节点，f: 节点u的父节点，确保深度优先搜索不会回到父节点上，d: 首都节点到u的距离

def dfs(u, f, d):
    r = d  # 目前为止找到的到u最远的节点的距离，以d作为初始值

    # 遍历从u出发的所有边
    for p in e[u]:
        if p[0] != f:
            # 考虑目前为止已经搜索到的最远距离，以及以p[0]为子树的节点中，到p[0]距离最远的节点，到p[0]的距离
            r = max(r, dfs(p[0], u, d + p[1]))
    return r

# 最终的答案，全部边长之和的2倍，减去从首都节点出发，到终结节点的距离。
print(s * 2 - dfs(1, 0, 0))
```

```python
# 这个题解使用bfs计算各个节点到首都节点的距离，并且没有使用递归，所以没有栈溢出的风险
from collections import deque

n = int(input())
e = [[] for _ in range(n + 1)]
s = 0
for i in range(n - 1):
    u, v, w = map(int, input().split())
    e[u].append((v, w))
    e[v].append((u, w))
    s += w

# 使用BFS找到从根节点（城市1）到最远节点的距离
dist = [0] * (n + 1)       # 存储每个节点到根节点的距离
parent = [0] * (n + 1)     # 记录父节点以防回溯
queue = deque()
queue.append(1)
max_dist = 0

while queue:
    u = queue.popleft()
    for v, w in e[u]:
        if v == parent[u]:  # 避免回溯到父节点
            continue
        parent[v] = u
        dist[v] = dist[u] + w
        if dist[v] > max_dist:
            max_dist = dist[v]
        queue.append(v)

print(2 * s - max_dist)
```