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

# Python 六级

2026 年 06 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 | D | B | C | A | A | B | A | A | B | B | D | B | A | B | A |

**第 1 题** 下列关于 Python 中继承和多态的描述中，错误的是（ ）。

- **A.** 当使用父类类型变量指向子类对象并调用方法时，会执行子类重写后的方法
- **B.** 子类通过重写父类中已有的方法，可以实现多态的效果
- **C.** Python 中不需要手动声明虚方法，普通方法默认就支持多态调用
- **D.** 构造方法 `__init__` 可以在运行时动态绑定，因此能像普通方法一样实现多态创建对象

**第 2 题** 下列代码中，执行 `d1.work()` 和 `d2.work()` 输出不同结果的主要原因是（ ）。

```python
class Device:
    def work(self):
        print("Device is working")

class Printer(Device):
    def work(self):
        print("Printer is printing")

class Scanner(Device):
    def work(self):
        print("Scanner is scanning")

if __name__ == '__main__':
    d1 = Printer()
    d2 = Scanner()

    d1.work()
    d2.work()
```

- **A.** `Printer` 和 `Scanner` 使用了相同的构造函数
- **B.** 子类重写了父类的 `work` 方法，Python 会根据对象实际类型调用对应版本的方法
- **C.** `d1` 和 `d2` 是不同的变量
- **D.** 程序中使用了 `del` 释放对象

**第 3 题** 下面代码在 `main()` 中有一行会导致错误，请找出来（ ）。

```python
class Student:
    def __init__(self, n, s):
        self.__name = n
        self.__score = s

    def get_name(self):
        return self.__name

    def set_score(self, s):
        self.__score = s


if __name__ == '__main__':
    stu = Student("Tom", 85)
    print(stu.get_name())    # ①
    stu.set_score(90)        # ②
    print(stu.__score)       # ③
    print(stu.get_name())    # ④
```

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

**第 4 题** 某文本编辑器把用户输入的字符依次压入栈 `S`。用户依次输入 `X, Y, Z, W` 后，连续执行两次撤销操作。每次撤销都会弹出栈顶一个字符。此时栈从栈底到栈顶的内容是（ ）。

- [ ] **A.** `X Y`
- [ ] **B.** `X Y Z`
- [ ] **C.** `Y Z`
- [ ] **D.** `X Z`

**第 5 题** 假设循环队列数组长度为 `N = 7`，队空判断条件为 `front == rear`。入队和出队操作如下：

```python
N = 7
q = [0] * N
front = 3
rear = 3

def enqueue(x):
    global rear
    q[rear] = x
    rear = (rear + 1) % N

def dequeue():
    global front
    front = (front + 1) % N
```

依次执行：

```
enqueue(10)
enqueue(20)
enqueue(30)
dequeue()
enqueue(40)
dequeue()
enqueue(50)
```

最终 `(front, rear)` 的值是（ ）。

- [ ] **A.** `(5, 1)`
- [ ] **B.** `(4, 0)`
- [ ] **C.** `(5, 0)`
- [ ] **D.** `(3, 1)`

**第 6 题** 以下函数 `check()` 用于判断一棵二叉树是否为（ ）。

```python
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def check(root):
    if not root:
        return True

    q = deque()
    q.append(root)
    has_null = False

    while q:
        cur = q.popleft()

        if not cur:
            has_null = True
        else:
            if has_null:
                return False
            q.append(cur.left)
            q.append(cur.right)

    return True
```

- [ ] **A.** 满二叉树
- [ ] **B.** 完全二叉树
- [ ] **C.** 二叉搜索树
- [ ] **D.** 平衡二叉树

**第 7 题** 以下代码实现了二叉树的哪种遍历方式？（ ）

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def traverse(root):
    if not root:
        return
    print(root.val, end=' ')
    traverse(root.left)
    traverse(root.right)
```

- [ ] **A.** 前序遍历
- [ ] **B.** 中序遍历
- [ ] **C.** 后序遍历
- [ ] **D.** 层序遍历

**第 8 题** 已知一棵二叉树的先序遍历序列为：`A B D E H C F G`，中序遍历序列为：`D B H E A F C G`，则该二叉树的后序遍历序列是（ ）。

- [ ] **A.** `D H E B F G C A`
- [ ] **B.** `D E H B F G C A`
- [ ] **C.** `H D E B F C G A`
- [ ] **D.** `D H E B G F C A`

**第 9 题** 有 6 个字符，它们出现的次数分别为：$\{3, 4, 7, 8, 12, 15\}$，现在用哈夫曼编码为这些字符编码，最小加权路径长度 `WPL` 的值为（ ）。

- [ ] **A.** 113
- [ ] **B.** 119
- [ ] **C.** 126
- [ ] **D.** 31

**第 10 题** 对 `n` 个不同符号进行哈夫曼编码。若生成的哈夫曼树共有 63 个结点，则 `n` 的值是（ ）。

- [ ] **A.** 31
- [ ] **B.** 32
- [ ] **C.** 63
- [ ] **D.** 64

**第 11 题** 在格雷码中，相邻两个编码只能有一位不同。若当前编码为 `110`，则它的下一个编码不可能是（ ）。

- [ ] **A.** `010`
- [ ] **B.** `111`
- [ ] **C.** `100`
- [ ] **D.** `001`

**第 12 题** 给定一棵二叉树，采用广度优先搜索 `BFS` 返回其右视图，其中右视图中的每个节点都是该层最右侧的节点。横线处应填写（ ）。

```python
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def rightSideView(root):
    result = []
    if not root:
        return result

    q = deque([root])

    while q:
        sz = len(q)
        for i in range(sz):
            node = q.popleft()

            ____________________________

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

    return result
```

- [ ] **A.** `if i == 0: result.append(node.val)`
- [ ] **B.** `if i == sz - 1: result.append(node.val)`
- [ ] **C.** `result.append(q[0].val)`
- [ ] **D.** `if node.right: result.append(node.right.val)`

**第 13 题** 下面代码实现二叉搜索树的插入操作。假设树中不存在重复值，横线处应填写（ ）。

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def insertNode(root, x):
    if not root:
        return TreeNode(x)

    if x < root.val:
        ____________________________
    else:
        root.right = insertNode(root.right, x)

    return root
```

- [ ] **A.** `root.left = insertNode(root.left, x)`
- [ ] **B.** `root = insertNode(root.left, x)`
- [ ] **C.** `root.right = insertNode(root.left, x)`
- [ ] **D.** `insertNode(root.left, x)`

**第 14 题** 给定一个整数数组 `a`，每个元素表示一个位置上的数值。要求从数组中选择若干个元素，使得任意两个被选择的元素在原数组中都不相邻，并且所选元素的总和最大。函数 `choose(a)` 返回能够得到的最大总和，则横线处应填写（ ）。

```
def choose(a):
    if not a:
        return 0

    n = len(a)
    if n == 1:
        return a[0]

    dp = [0] * n
    dp[0] = a[0]
    dp[1] = max(a[0], a[1])

    for i in range(2, n):
        dp[i] = ____________________________

    return dp[-1]
```

- A. `dp[i - 1] + a[i]`
- B. `max(dp[i - 1], dp[i - 2] + a[i])`
- C. `max(dp[i - 2], a[i])`
- D. `dp[i - 1] + dp[i - 2]`

**第 15 题** 下面代码实现 0/1 背包的一维动态规划。第 `i` 个物品重量为 `wt[i]`，价值为 `val[i]`，背包容量为 `W`。横线处应填写（ ）。

```
def knapsack(W, wt, val):
    n = len(wt)
    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[w - wt[i]] + val[i])`
- B. `dp[w] = max(dp[w - 1], dp[w - wt[i]] + val[i])`
- C. `dp[w] = dp[w] + val[i]`
- D. `dp[w - wt[i]] = max(dp[w], 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** Python 中没有虚方法的概念，`__init__` 也不能声明为虚方法，且构造方法不实现运行时多态。

**第 2 题** 在 Python 中，执行 `del` 删除一个子类对象时，会自动先调用子类的 `__del__` 方法，再调用父类的 `__del__` 方法。

**第 3 题** 在 Python 标准库中，`list` 模拟栈时，`list` 非空的场景下，`pop()` 函数不传参数被调用时，会返回栈顶元素并将其删除。

**第 4 题** 程序运行后会输出 `2`。

```python
from collections import deque

q = deque()
q.append(1)
q.append(2)
q.append(3)
q.popleft()
print(q[0])
```

**第 5 题** 以下函数可以正确完成二叉搜索树的插入，并保持二叉搜索树性质。

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

def insertNode(root, x):
    if not root:
        return TreeNode(x)

    if x < root.val:
        root.right = insertNode(root.right, x)
    else:
        root.left = insertNode(root.left, x)

    return root
```

**第 6 题** 哈夫曼编码一定唯一，只要字符频率相同，得到的编码也一定完全相同。

**第 7 题** 若用数组按层序存储完全二叉树，且根节点下标为 `0`，则下标为 `i` 的节点左孩子下标为 `2 * i + 1`，右孩子下标为 `2 * i + 2`。

**第 8 题** 以下代码可以正确地按层换行输出二叉树的节点值。

```python
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def printByLevel(root):
    if not root:
        return

    q = deque([root])

    while q:
        for i in range(len(q)):
            cur = q.popleft()
            print(cur.val, end=' ')

            if cur.left:
                q.append(cur.left)
            if cur.right:
                q.append(cur.right)
        print()
```

**第 9 题** 使用栈非递归实现二叉树前序遍历时，若希望先访问左子树，通常应先将右孩子入栈，再将左孩子入栈。

**第 10 题** 动态规划问题通常要求具有最优子结构，并且常常存在重叠子问题。

# 3 编程题（每题 25 分，共 50 分）

## 3.1 编程题 1

- **试题名称**：条形蛋糕
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

### 3.1.1 题目描述

寒假到了，小杨同学打算找一份兼职，顺便体验一下打工人的生活。

小杨同学给一家蛋糕店发送了一份自己的简历，希望可以在寒假来这里帮忙。店长最近正好遇到了一个难题：店里每天会做一条长条蛋糕，但是不同长度的蛋糕块卖出的价格不同，应该怎么分才能卖得最多呢?

有趣的是店长曾经学习过计算机专业。他最近对动态规划算法很感兴趣，于是打算用这个问题考一考小杨同学，问题如下：

- 给定一条长度为 $n$ 的长条蛋糕和一个价格表，该价格表表示长度为 $i$（$i = 1, 2, \ldots, n$）的蛋糕块的价格为 $p_i$。求蛋糕的分割方案，使得总销售价格最大，注意**蛋糕块的长度必须为整数**。

### 3.1.2 输入格式

第一行一个正整数 $n$（$1 \leq n \leq 10^3$），表示长条蛋糕的总长度。

第二行 $n$ 个正整数 $p_1, p_2, \ldots, p_n$（$1 \leq p_i \leq 10^5$），表示不同长度蛋糕块的价格。

### 3.1.3 输出格式

一行一个正整数，表示最大总销售价格。

### 3.1.4 输入样例 1

```
4
1 5 8 9
```

### 3.1.5 输出样例 1

```
10
```

### 3.1.6 输入样例 2

```
10
1 5 8 9 10 17 17 20 24 30
```

### 3.1.7 输出样例 2

```
30
```

### 3.1.8 样例解释

第一个样例中，长度为 1 的蛋糕价值为 1，长度为 2 的蛋糕价值为 5，长度为 3 的蛋糕价值为 8，长度为 4 的蛋糕价值为 9；

总长度为 4 的长条蛋糕，有 $\{4\},\{1,3\},\{2,2\},\{1,1,2\},\{1,1,1,1\}$ 五种本质不同的分法。

其对应的总销售价格分别为 $9,9,10,7,4$，故最大总销售价格为 10。

第二个样例中，长度为 10 的长条蛋糕，销售价格最大的分法为 $\{10\}$，最大总销售价格为 30。

### 3.1.9 参考程序 1

```python
# 蛋糕长度
n = int(input())
# 蛋糕长度及其对应的价钱
p = list(map(int, input().split()))
p = [0] + p
# f[i][j] = 只使用长度 1..i 的蛋糕块，总长度为 j 时的最大总价
f = [[0] * (n + 1) for _ in range(n + 1)]
for i in range(1, n + 1):
    for j in range(1, n + 1):
        # 判断是否要切一块长度为i的蛋糕更好，f[i][j-i]+p[i]表示最后一块蛋糕长度为i，前者是原有方案
        if j >= i:
            f[i][j] = max(f[i - 1][j], f[i][j - i] + p[i])
        # 如果j比i小，那么没法切长度为i的蛋糕了
        else:
            f[i][j] = f[i - 1][j]

print(f[n][n])
```

### 3.1.10 参考程序 2

```python
# 蛋糕长度
n = int(input())
# 蛋糕长度及其对应的价钱
p = list(map(int, input().split()))
p = [0] + p
# f[i][j] = 只使用长度 1..i 的蛋糕块，总长度为 j 时的最大总价，但是i维度被隐藏了。这个版本耗费的空间更小，时间复杂度一样
f = [0] * (n + 1)
# 当前最大的蛋糕长度
for i in range(1, n + 1):
    # 蛋糕长度j
    for j in range(i, n + 1):
        # 长度为j，且最大分块不超过i的蛋糕的价钱，要么是原有的方案，要么是最后一刀为i的方案。
        # 需要注意的是f[j-i]可能其最优价格已经包含了长度为i的蛋糕，因此f[j]不一定只包含一块长度为i的蛋糕
        f[j] = max(f[j], f[j - i] + p[i])
print(f[n])
```

## 3.2 编程题 2

- **试题名称**：满二叉树
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

### 3.2.1 题目描述

给定一棵包含 $n$ 个结点的**有根二叉树**，结点依次以 $1, 2, \ldots, n$ 编号，根结点编号为 1。

对于结点 $i$，其左儿子的编号记为 $l_i$，右儿子编号记为 $r_i$。特别地，如果左儿子不存在则 $l_i = 0$，如果右儿子不存在则 $r_i = 0$。

树中每个结点都对应一棵以其为根的子树。请你求出给定有根树的所有 $n$ 棵子树中，有多少棵子树是满二叉树。

满二叉树是指所有叶子深度均相同，且除叶子外均有两个儿子的二叉树，例如以下三棵二叉树均是满二叉树：

```
()      ()            ()
       /  \          /  \
     ()    ()      ()    ()
                  /  \  /  \
                ()   ()()   ()
```

例如，

```
   ()
  /  \
()    ()
```

在上面这棵有 3 个节点的二叉树中，有 3 个子树是满二叉树（包括整个树本身，及所有的单个叶子节点）；

又例如，

```
   (1)
  /   \
(2)   (3)
     /   \
   (4)   (5)
```

在上面这棵有 5 个节点的二叉树中，有 4 个子树是满二叉树（包括节点 3 的子树，以及所有单个叶子节点）。

### 3.2.2 输入格式

第一行，一个正整数 $n$，表示有根二叉树结点数量。

接下来 $n$ 行，每行两个非负整数 $l_i, r_i$，表示结点 $i$ 的左儿子编号和右儿子编号，整数之间以空格分隔。

### 3.2.3 输出格式

输出一行，一个整数，表示所有子树中满二叉树的数量。

### 3.2.4 样例

### 3.2.5 输入样例 1

```
4
2 3
4 0
0 0
0 0
```

### 3.2.6 输出样例 1

```
2
```

### 3.2.7 输入样例 2

```
3
2 3
0 0
0 0
```

### 3.2.8 输出样例 2

```
3
```

### 3.2.9 数据范围

对于 $40\%$ 的测试点，保证 $1 \leq n \leq 500$。

对于所有测试点，保证 $1 \leq n \leq 10^5$。

### 3.2.10 参考程序

```python
# 结点数量
n = int(input())
# s[i] = (li, ri), i 的左/右儿子编号, 0 表示不存在
# 所有节点从1开始编号, 所以不用管0号
s = [[]] + [list(map(int, input().split())) for _ in range(n)]

# BFS 获得层序遍历, 目的是后续倒序处理时保证子女先于父母
# q: 队列, 最初只有节点1, 不断扩展将后续的节点放进来得到BFS遍历序列, 越深的节点越靠后
# ql: 当前需要扩展哪个节点的子节点进入q, 初始下标为0
q, ql = [1], 0
while ql < len(q):
    q.extend([u for u in s[q[ql]] if u])
    ql += 1

# h[i]: 以 i 为根的子树高度 (叶子高度为 1)
# sz[i]: 以 i 为根的子树结点数
h, sz = [0] * (n + 1), [0] * (n + 1)
ans = 0
# 倒序 BFS 列表 = 自底向上遍历
# 遍历节点u, 这个时候节点u的子节点的高度和节点数已经统计完了
for u in q[::-1]:
    l, r = s[u]
    # 以u为根的树的高度
    h[u] = max(h[l], h[r]) + 1
    # 以u为根的树的节点数
    sz[u] = sz[l] + sz[r] + 1
    # 满二叉树结点数 = 2^h - 1; 等价判断 sz + 1 == 2^h
    if sz[u] + 1 == 2 ** h[u]:
        ans += 1
print(ans)
```