



# Python 六级

2026 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	C	C	D	A	D	B	A	C	D	B	B	B	C	C	A

第 1 题 以下关于 Python 类继承的代码，执行后输出结果是？（）

```
1 class Animal:
2     def __init__(self, name):
3         self.name = name
4     def speak(self):
5         return "动物叫声"
6
7 class Dog(Animal):
8     def speak(self):
9         return f"{self.name}: 汪汪汪"
10
11 class Cat(Animal):
12     def __init__(self, name, color):
13         super().__init__(name) # 调用父类构造方法
14         self.color = color
15     def speak(self):
16         return f"{self.name} ({self.color}) : 喵喵喵"
17
18 dog = Dog("旺财")
19 cat = Cat("咪宝", "橘色")
20 print(dog.speak(), cat.speak())
```

- A. 动物叫声 动物叫声
- B. 旺财: 汪汪汪 咪宝: 喵喵喵
- C. 旺财: 汪汪汪 咪宝 (橘色) : 喵喵喵
- D. 动物叫声 咪宝 (橘色) : 喵喵喵

第 2 题 下列代码中，s1.draw() 和 s2.draw() 能正确运行并输出不同结果的主要原因是（）。

```

1 class Shape:
2     def draw(self):
3         print("绘制图形")
4
5 class Circle(Shape):
6     def draw(self):
7         print("绘制圆形")
8
9 class Rectangle(Shape):
10    def draw(self):
11        print("绘制矩形")
12
13 if __name__ == "__main__":
14     s1 = Circle()
15     s2 = Rectangle()
16
17     s1.draw()
18     s2.draw()

```

- A. draw() 是普通成员函数
- B. Shape 中的 draw() 被声明为虚函数
- C. Circle 和 Rectangle 中使用了公有继承
- D. 对象变量名不同

第3题 下面的代码在主程序 `if __name__ == "__main__":` 中有没有一行会导致运行错误，如果有请找出错误行()

```

1 class Pet:
2     def __init__(self, n, a):
3         self._name = n
4         self._age = a
5     def get_name(self):
6         return self._name
7     def birthday(self):
8         self._age += 1
9
10 if __name__ == "__main__":
11     cat = Pet("奶茶", 2)
12     print(cat.get_name()) # ①
13     cat.birthday() # ②
14     cat._name = "大橘" # ③
15     print(cat.get_name()) # ④

```

- A. 第①行
- B. 第②行
- C. 第③行
- D. 无错误行

第4题 游乐园的过山车每次限坐 4 人，用循环队列管理排队（容量 MAX=5，空一格判满）。下面代码执行后，循环队列是否已满？rear 的值是多少？

```

1 MAX = 5
2 queue = [None] * MAX
3 front = 0
4 rear = 0
5 def enqueue(x):
6     global rear, queue
7     queue[rear] = x
8     rear = (rear + 1) % MAX
9
10 def dequeue():
11     global front
12     front = (front + 1) % MAX
13
14 if __name__ == "__main__":
15     enqueue(1)
16     enqueue(2)
17     enqueue(3)
18     enqueue(4)
19
20     dequeue()
21     dequeue()
22
23     enqueue(5)
24     enqueue(6)
25
26     print("队列数组: ", queue)
27     print("front下标位置: ", front)
28     print("rear下标位置: ", rear)

```

- A. 已满, rear = 1
- B. 未满, rear = 1
- C. 已满, rear = 2
- D. 未满, rear = 4

第5题 在以下计算机系统应用场景中, 最适合使用循环队列的是 ( )。

- A. 函数调用过程中, 保存局部变量和返回地址
- B. 表达式求值中的运算符优先级处理
- C. 操作系统中的进程优先级调度 (高优先级先执行)
- D. 生产者和消费者问题中的共享缓冲区

第6题 在二叉搜索树 (BST) 中, 若中序遍历的序列为 {1, 2, 3, 4, 5}, 且先序遍历的第一个序列元素为 3, 则下列说法正确的是 ( )。

- A. 该树一定是一棵完全二叉树。
- B. 元素4和5不可能是兄弟节点。
- C. 元素1所在节点的深度可能大于3 (根节点深度为1)。
- D. 元素2一定是元素1的父节点。

第7题 某二叉树共有10个结点, 记为A~J, 已知它的先序遍历序列为: **ABDHIECFJG**, 中序遍历序列为: **HDIBEA FJCG**, 则该二叉树的后序遍历序列是 ( )。

- A. HIDEBJFGCA
- B. HIDBEJFGCA
- C. IHDEBJFGCA

D. H I D E B F J G C A

第 8 题 下列关于树的遍历的说法中，正确的一项是（ ）。

- A. 对任意一棵树进行深度优先遍历，所得序列一定唯一。
- B. 已知一棵二叉树的先序遍历和后序遍历序列，可以唯一确定这棵二叉树。
- C. 若一棵二叉树的先序遍历序列与中序遍历序列相同，则该二叉树一定为只有右子树的链式结构。
- D. 已知一棵二叉树的先序遍历序列，即可唯一地确定该二叉树的结构。

第 9 题 有 6 个字符，它们出现的次数分别为：{2, 3, 3, 4, 6, 8}，现在用哈夫曼编码为这些字符编码，最小加权路径长度 WPL（每个字符的出现次数×它的编码长度，再把每个字符结果加起来）的值为（ ）。

- A. 58
- B. 60
- C. 62
- D. 64

第 10 题 对  $n$  个不同符号的符号进行哈夫曼编码。若生成的哈夫曼树共有 115 个结点，则  $n$  的值是（ ）。

- A. 60
- B. 58
- C. 57
- D. 56

第 11 题 关于格雷编码（Gray Code），下列说法正确的是（ ）。

- A. 格雷编码中，编码位数越多，相邻编码之间变化的位数也越多
- B. 格雷编码中，相邻两个编码的二进制位恰好有一位不同
- C. 格雷编码就是把普通二进制编码按位取反后得到的结果
- D. 格雷编码不能用于数字电路和状态转换的设计中

第 12 题 给定一棵二叉树，采用广度优先搜索 (BFS) 算法，返回右视图所有节点的值。其中右视图定义为：二叉树的右视图是从树的右侧看过去时可见的节点集合，即右视图中的每个节点都是某一层中最右侧的节点。

```

1 class TreeNode:
2     def __init__(self, x):
3         self.val = x
4         self.left = None
5         self.right = None
6
7 def rightSideView(root):
8     rightmost_value_at_depth = {}
9     max_depth = -1
10
11     node_queue = []
12     depth_queue = []
13     node_queue.append(root)
14     depth_queue.append(0)
15
16     while node_queue:
17         node = node_queue.pop(0)
18         depth = depth_queue.pop(0)
19
20         if node is not None:
21             max_depth = max(max_depth, depth)
22             rightmost_value_at_depth[depth] = node.val
23
24             node_queue.append(node.left)
25             node_queue.append(node.right)
26
27             # (1)
28             depth_queue.append(_____)
29             depth_queue.append(_____)
30
31
32     right_view = []
33     # (2) 补全
34     for depth in range(_____):
35         right_view.append(rightmost_value_at_depth[depth])
36
37     return right_view
38
39 if __name__ == "__main__":
40     root = TreeNode(1)
41     root.left = TreeNode(2)
42     root.right = TreeNode(3)
43     root.left.right = TreeNode(5)
44     root.right.right = TreeNode(4)
45
46     print(rightSideView(root))

```

- A. (1) depth, depth; (2) 0, max\_depth
- B. (1) depth + 1, depth + 1; (2) 0, max\_depth + 1
- C. (1) depth + 1, depth + 1; (2) 1, max\_depth + 1
- D. (1) depth, depth; (2) 1, max\_depth

第 13 题 下列关于树的深度优先搜索 (DFS) 的说法中, 正确的是 ( )。

- A. 对树进行 DFS 时, 一定是按层从上到下依次访问结点
- B. 对任意一棵树进行 DFS, 得到的遍历序列唯一
- C. 对一棵树进行 DFS 时, 常借助递归或栈实现
- D. DFS 只能用于二叉树, 不能用于普通树

第 14 题 小朋友们去邻里拜年，每个家里有不同数量的糖果。规则是：不能连续进入两个相邻的房子（即不能同时取相邻两家的糖果）。目标是拿到最多糖果。以下是代码实现，请补全横线。

```

1  def visit(nums):
2      if not nums:
3          return 0
4
5      size = len(nums)
6      if size == 1:
7          return nums[0]
8
9      dp = [0] * size
10     dp[0] = nums[0]
11     dp[1] = max(nums[0], nums[1])
12
13     for i in range(2, size):
14         -----
15
16     return dp[size - 1]
17 if __name__ == "__main__":
18     nums1 = [1,2,3,1]
19     print(visit(nums1))
20
21     nums2 = [2,7,9,3,1]
22     print(visit(nums2))
23
24     nums3 = [5]
25     print(visit(nums3))
26
27     nums4 = []
28     print(visit(nums4))

```

- A.  $dp[i] = dp[i - 1] + nums[i]$
- B.  $dp[i] = \max(dp[i - 1], dp[i - 2] * nums[i])$
- C.  $dp[i] = \max(dp[i - 1], dp[i - 2] + nums[i])$
- D.  $dp[i] = dp[i - 2] + nums[i]$

第 15 题 元宵节晚上，小朋友沿着一条发光石板路前进，每次可以向前走 1 块或 2 块石板。若  $dp[i] = dp[i - 1] + dp[i - 2]$ ，下面关于  $dp[i]$  的含义最合适的是（ ）。

- A. 走到第  $i$  块石板的不同走法数量
- B. 走到第  $i$  块石板时，已经走过的石板总数
- C. 从第  $i$  块石板走回起点的最少步数
- D. 从第  $i$  块石板走回起点的最大步数

## 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 下面定义了一个表示二维坐标点的类 Point，并提供了一个带参数的构造函数，但第 ② 行 Point b; 会调用编译器自动生成的默认构造函数，将 b.x 和 b.y 被初始化为 0.0，程序可以正常编译运行。

```

1 class Point:
2     def __init__(self, px, py):
3         self.x = px
4         self.y = py
5
6     def print(self):
7         print(f"({self.x}, {self.y})")
8
9 if __name__ == "__main__":
10    a = Point(3.0, 4.0) # ①
11    # ②
12    b = Point()
13    a.print()

```

第 2 题 Python 中的继承支持单继承和多继承，但子类无法直接访问父类的私有成员。

第 3 题 对如下结构的树，执行 `travel` 函数，输出结果是 1 2 3 4 5 。

```

1      1
2     / \
3    2   3
4   / \
5  4   5

```

```

1 class Node:
2     def __init__(self, v):
3         self.val = v
4         self.left = None
5         self.right = None
6
7 def travel(root):
8     if not root:
9         return
10    stack = []
11    stack.append(root)
12
13    while stack: =
14        cur = stack.pop()
15        print(cur.val, end=" ")
16        if cur.right:
17            stack.append(cur.right)
18        if cur.left:
19            stack.append(cur.left)
20
21 if __name__ == "__main__":
22
23    root = Node(1)
24    root.left = Node(2)
25    root.right = Node(3)
26    root.left.left = Node(4)
27    root.left.right = Node(5)
28    travel(root)

```

第 4 题 若所有字符出现频率相同，则哈夫曼编码一定会得到完全二叉树。

第 5 题 哈夫曼编码是一种变长的前缀编码，在解码时不需要额外的分隔符就能唯一还原，这是因为在哈夫曼树中，任何一个字符的叶子结点都不会成为另一个字符结点的祖先。

第 6 题 在 Python 中使用列表存储按层序遍历的完全二叉树时，若根节点存储在 `tree[0]`，则对于任意非空节点 `tree[i]`，其右孩子（如果存在）必然位于 `tree[2 * i + 2]`。

**第7题** 在 Python 中使用列表模拟栈来非递归地实现二叉树的前序遍历，为了保证遍历顺序正确，在处理完当前结点后，应该先将该结点的左孩子压入栈中，然后再将右孩子压入栈中。（）

**第8题** 设二叉树共有  $n$  个结点，函数 `preorderTraversal` 的时间复杂度为  $O(n)$ ，空间复杂度为  $O(n)$ 。

```
1 class TreeNode:
2     def __init__(self, x):
3         self.val = x
4         self.left = None
5         self.right = None
6
7 def preorder(root, res):
8     if root is None:
9         return
10    res.append(root.val)
11    preorder(root.left, res)
12    preorder(root.right, res)
13 def preorderTraversal(root):
14    res = []
15    preorder(root, res)
16    return res
17
18
19 if __name__ == "__main__":
20
21    root = TreeNode(1)
22    root.right = TreeNode(2)
23    root.right.left = TreeNode(3)
24    result = preorderTraversal(root)
25    print("前序遍历结果: ", result)
```

**第9题** 以下代码实现了0-1背包问题的一维动态规划解法，内层循环采用经典的逆序遍历方式。若将内层循环改为正序遍历（即 `for j in range(w[i], W + 1):`），仍能得到正确答案。

```
1 def knapsack_01():
2     W = 5
3     w = [2, 3, 4]
4     v = [10, 1, 1]
5     n = 3
6     dp = [0] * (W + 1)
7
8     for i in range(n):
9         for j in range(W, w[i] - 1, -1):
10            dp[j] = max(dp[j], dp[j - w[i]] + v[i])
11
12    print(dp[W])
13
14 knapsack_01()
```

**第10题** 在动态规划问题中，状态空间相同且没有重复计算的情况下，“状态转移方程+递推”与“递归+记忆化搜索”的时间复杂度通常相同。

### 3 编程题（每题 25 分，共 50 分）

#### 3.1 编程题 1

- 试题名称：选数
- 时间限制：1.0 s
- 内存限制：512.0 MB

##### 3.1.1 题目描述

给定两个包含  $n$  个整数的数组  $a = [a_1, \dots, a_n]$  与  $b = [b_1, \dots, b_n]$ 。你需要指定若干下标  $p_1 < \dots < p_k$  ( $1 \leq k \leq n$ ) 使得以下条件成立：

- $1 \leq p_i \leq n$  ( $1 \leq i \leq k$ ) ；
- $p_{i+1} \geq p_i + b_{p_i}$  ( $1 \leq i < k$ ) 。

你需要在满足以上条件的前提下最大化  $\sum_{i=1}^k a_{p_i}$ ，也即最大化数组  $a$  对应下标的整数之和。

##### 3.1.2 输入格式

第一行，一个正整数  $n$ ，表示数组长度。

第二行， $n$  个非负整数  $a_1, a_2, \dots, a_n$ ，表示数组  $a$ 。

第三行， $n$  个非负整数  $b_1, b_2, \dots, b_n$ ，表示数组  $b$ 。

##### 3.1.3 输出格式

一行，一个整数，表示在满足下标条件的前提下，数组  $a$  对应下标的整数之和的最大值。

##### 3.1.4 样例

###### 3.1.4.1 输入样例 1

```
1 | 4
2 | 1 2 3 4
3 | 3 3 1 1
```

###### 3.1.4.2 输出样例 1

```
1 | 7
```

###### 3.1.4.3 输入样例 2

```
1 | 6
2 | 1 1 4 5 1 4
3 | 1 2 3 2 1 0
```

###### 3.1.4.4 输出样例 2

```
1 | 11
```

### 3.1.5 数据范围

对于 40% 的测试点，保证  $2 \leq n \leq 10^3$ 。

对于所有测试点，保证  $2 \leq n \leq 10^5$ ， $0 \leq a_i \leq 10^9$ ， $0 \leq b_i \leq n$ 。

### 3.1.6 参考程序

```
1 # 接受输入
2 # n
3 n = int(input())
4 # 接受a和b的输入
5 a = [0] + list(map(int, input().split()))
6 b = [0] + list(map(int, input().split()))
7
8 # 准备动态规划的数组f
9 # f[i]的含义如题解所述
10 f = [0] * (n + 5)
11 # ans表示目前找到的最大的答案
12 ans = 0
13
14 for i in range(1, n + 1):
15     # 比较一下，如果将i加入到下标序列中，那么可能得到的最大序列和为f[i]+a[i]，比较这个值和最大的已发现的答
    案
16     ans = max(ans, f[i] + a[i])
17
18     # f[i+b[i]]表示最后一个被选中的下标<=i的情况下能够找到的最大序列和。如果加入a[i]能够让这个值更大，则
    更新
19     if i + b[i] <= n:
20         f[i + b[i]] = max(f[i + b[i]], f[i] + a[i])
21     # 因为f[i]表示的是i及i以后的下标都能被选中情况下的最大序列和，所以利用f[i]来更新f[i+1]
22     f[i + 1] = max(f[i + 1], f[i])
23
24 print(ans)
```

## 3.2 编程题 2

- 试题名称：完全二叉树
- 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  $1, 2, \dots, n$  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，有多少棵子树是完全二叉树。

### 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，每行两个正整数  $l_i, r_i$ ，表示结点  $i$  的左儿子编号和右儿子编号。

### 3.2.3 输出格式

输出一行，一个整数，表示所有子树中完全二叉树的数量。

### 3.2.4 样例

#### 3.2.4.1 输入样例 1

```
1 | 4
2 | 2 3
3 | 4 0
4 | 0 0
5 | 0 0
```

#### 3.2.4.2 输出样例 1

```
1 | 4
```

#### 3.2.4.3 输入样例 2

```
1 | 4
2 | 2 3
3 | 0 0
4 | 4 0
5 | 0 0
```

#### 3.2.4.4 输出样例 2

```
1 | 3
```

### 3.2.5 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  $1 \leq n \leq 10^5$ 。

### 3.2.6 参考程序

```
1 n = int(input())
2 # 左子节点编号
3 l = [0] * (n + 1)
4 # 右子节点编号
5 r = [0] * (n + 1)
6 # 节点的“最小叶子节点深度”
7 mn = [0] * (n + 1)
8 # 节点的深度
9 mx = [0] * (n + 1)
10 # 以节点i为根的子树是否为完全二叉树, 1表示是, 0表示不是
11 chk = [0] * (n + 1)
12
13 ans = 0
14
15 for i in range(1, n + 1):
16     li, ri = map(int, input().split())
17     l[i] = li
18     r[i] = ri
19
20 def dfs(u):
21     global ans
22     # 先假设他是完全二叉树
23     chk[u] = 1
24     if u == 0:
25         return
26     # 后序遍历。先遍历左子树, 再遍历右子树
27     dfs(l[u])
28     dfs(r[u])
29     # 完全二叉树左右子树必须都是完全二叉树
30     chk[u] &= chk[l[u]] & chk[r[u]]
31     # 节点u的“最小叶子节点深度”为左右子树“最小叶子节点深度”中较小的+1
32     mn[u] = 1 + min(mn[l[u]], mn[r[u]])
33     # 节点u的深度为左右子树深度较大的+1
34     mx[u] = 1 + max(mx[l[u]], mx[r[u]])
35
36     # 左子树的“最小叶子节点深度”必定大于等于右子树的深度
37     chk[u] &= mn[l[u]] >= mx[r[u]]
38     # 满深度必定大于等于深度-1
39     chk[u] &= mn[u] >= mx[u] - 1
40
41     ans += chk[u]
42
43 dfs(1)
44
45 print(ans)
```