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

Python 五级

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

第 1 题 小杨使用单向链表存放任务序列，同时记录链表的头部结点 `head` 和尾部结点 `tail`。链表不为空，已知尾部结点 `tail`，在链表末尾添加新结点的时间复杂度是（　　）。

```
class Node:
    def __init__(self, val):
        self.value = val
        self.next = None

head: Node | None
tail: Node | None
```

- A. $O(1)$
- B. $O(\log n)$
- C. $O(n)$
- D. $O(n \log n)$

第 2 题 在不带哨兵结点的双向链表中，结点 `p` 既不是头部结点，也不是尾部结点。删除结点 `p` 的正确代码是（　　）。

```
class Node:
    def __init__(self, val):
        self.value = val
        self.prev = None   # 指向前面结点
        self.next = None   # 指向后面结点
```

- A.

```
p.prev.next = p.next
p.next.prev = p.prev
```

- B.

```
p.prev = p.prev.next
p.next = p.next.prev
```

- C.

```
p.prev.next = p.prev
p.next.prev = p.next
```

- [ ] **D.**

```
p.next = p.next.next
p.prev = p.prev.prev
```

第 3 题 下面函数使用快慢变量查找单向链表的中间结点。横线处应填写（　　）。

```
class Node:
    def __init__(self, val):
        self.value = val
        self.next = None

def middle(head):
    slow = head
    fast = head
    while fast is not None and fast.next is not None:
        slow = slow.next
        _____________
    return slow
```

- [ ] **A.** `fast = fast.next`
- [ ] **B.** `fast = fast.next.next`
- [ ] **C.** `fast = slow.next`
- [ ] **D.** `fast = slow.next.next`

第 4 题 函数 `gcd(a, b)` 定义如下，则 `gcd(105, 45)` 的结果是（　　）。

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

- [ ] **A.** `3`
- [ ] **B.** `5`
- [ ] **C.** `15`
- [ ] **D.** `45`

第 5 题 下面函数用于判断正整数 `n` 是否为质数。横线处的最佳写法是（　　）。

```
def isPrime(n):
    if n < 2:
        return False
    i = 2
    while _____________:
        if n % i == 0:
            return False
        i += 1
    return True
```

- [ ] **A.** `i < n`
- [ ] **B.** `i <= n // 2`
- [ ] **C.** `i * i < n`
- [ ] **D.** `i * i <= n`

第 6 题 下面代码实现线性筛（欧拉筛）。为保证每个合数只被它的最小质因子筛除一次，横线处应填写（　　）。

```
def linear_sieve(n):
    composite = [False] * (n + 1)
    primes = []
    for i in range(2, n + 1):
        if not composite[i]:
            primes.append(i)
        for p in primes:
            if i * p > n:
                break
            composite[i * p] = True
            if _____________:
                break
    return primes
```

- A. `p % i == 0`
- B. `i % p == 0`
- C. `i == p`
- D. `i * p == n`

第 7 题 根据唯一分解定理，整数 `756` 的正确质因数分解是（　　）。

- A. $2^2 \times 3^3 \times 7$
- B. $2^3 \times 3^2 \times 7$
- C. $2^2 \times 3^2 \times 21$
- D. $2 \times 3^3 \times 14$

第 8 题 函数 `f(n)` 定义如下，则 `f(4)` 的结果是（　　）。

```
def f(n):
    if n == 1:
        return 1
    return n + f(n - 1)
```

- A. `4`
- B. `7`
- C. `9`
- D. `10`

第 9 题 在升序数组中查找第一个严格大于 `x` 的元素下标，下面代码横线处应填写（　　）。

```
def upper_bound(a, x):
    l = 0
    r = len(a)
    while l < r:
        mid = l + (r - l) // 2
        if _____________:
            l = mid + 1
        else:
            r = mid
    return l
```

- A. `a[mid] < x`

- [ ] **B.** `a[mid] >= x`
- [ ] **C.** `a[mid] <= x`
- [ ] **D.** `a[mid] > x`

第 10 题 小杨需要把若干箱货物按原有顺序分配到 `days` 天中，每天运输连续的若干箱，求可以完成任务的最小载重量。函数 `check(cap)` 判断载重量为 `cap` 时，是否可以在规定天数运完全部货物。横线处应填写（　　）

```
l = maxWeight
r = totalWeight

while l < r:
    mid = l + (r - l) // 2
    if check(mid):
        _______________
    else:
        _______________
print(l)
```

- [ ] **A.** `l = mid + 1` 和 `r = mid`
- [ ] **B.** `r = mid` 和 `l = mid + 1`
- [ ] **C.** `r = mid - 1` 和 `l = mid`
- [ ] **D.** `l = mid` 和 `r = mid - 1`

第 11 题 下面是归并排序合并两个有序区间（升序排序）的片段。想要让排序算法保持稳定，横线处应当填写（　　）

```
while i <= mid and j <= right:
    if _______________:
        temp.append(a[i])
        i += 1
    else:
        temp.append(a[j])
        j += 1
```

- [ ] **A.** `a[i] < a[j]`
- [ ] **B.** `a[i] > a[j]`
- [ ] **C.** `a[i] >= a[j]`
- [ ] **D.** `a[i] <= a[j]`

第 12 题 下面快速排序的划分函数以 `a[right]` 作为枢轴，把不大于枢轴的元素移动到数组左侧。横线处应填写（　　）。

```
def partition(a, left, right):
    pivot = a[right]
    i = left - 1
    for j in range(left, right):
        if _______________:
            i += 1
            a[i], a[j] = a[j], a[i]
    a[i + 1], a[right] = a[right], a[i + 1]
    return i + 1
```

- [ ] **A.** `a[j] <= pivot`
- [ ] **B.** `a[j] > pivot`

- [ ] **C.** `a[i] <= pivot`
- [ ] **D.** `a[right] < a[j]`

第 13 题 小杨要在一间教室安排尽可能多场活动，每场活动拥有开始时间 `start` 和结束时间 `end` 。使用贪心算法，正确的选择策略是（　　）

```python
class Activity:
    def __init__(self, s, e):
        self.start = s
        self.end = e
```

- [ ] **A.** 每次选择开始时间最早的活动
- [ ] **B.** 每次选择持续时间最短的活动
- [ ] **C.** 每次选择参与人数最少的活动
- [ ] **D.** 按结束时间从早到晚排序，依次选择与已选活动不冲突的活动

第 14 题 下面函数用迭代方式求解最大连续子段和。对于数组 `[-2, 3, -1, 5, -6, 2]` ，函数返回结果是（　　）

```python
def maxSubArray(a):
    best = a[0]
    current = a[0]
    for i in range(1, len(a)):
        current = max(a[i], current + a[i])
        best = max(best, current)
    return best
```

- [ ] **A.** `5`
- [ ] **B.** `6`
- [ ] **C.** `7`
- [ ] **D.** `8`

第 15 题 数组 `a` 和 `b` 低位在前存储两个非负大整数，代码实现高精度加法，横线处应填写（　　）

```python
def add(a, b):
    c = []
    carry = 0
    n = max(len(a), len(b))
    for i in range(n):
        sum_val = carry
        if i < len(a):
            sum_val += a[i]
        if i < len(b):
            sum_val += b[i]
        c.append(sum_val % 10)
        ___________
    if carry != 0:
        c.append(carry)
    return c
```

- [ ] **A.** `carry = sum_val % 10`
- [ ] **B.** `carry = sum_val`
- [ ] **C.** `carry = sum_val // 10`
- [ ] **D.** `carry = c[i] // 10`

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 下面代码，在已知结点 `p` 的前提下，可以用 $O(1)$ 的时间，在单向链表的 `p` 结点后面插入新结点 `s`。

```
s.next = p.next
p.next = s
```

第 2 题 下面代码可以安全删除单向链表的头结点，并让 `head` 指向删除之后的新头结点。

```
p = head
head = p.next
```

第 3 题 下面的欧几里得算法，只要 `a`、`b` 是正整数，无论 `a > b` 还是 `a < b` 都可以正常求出最大公约数。

```
def gcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

第 4 题 下面埃氏筛从 `i * i` 开始标记合数，原因是：`i*i` 之前 `i` 的合数倍数，已经被更小的质因子标记完成。

```
def sieve(n):
    isPrime = [True] * (n + 1)
    isPrime[0] = isPrime[1] = False
    i = 2
    while i * i <= n:
        if isPrime[i]:
            j = i * i
            while j <= n:
                isPrime[j] = False
                j += i
        i += 1
    return isPrime
```

第 5 题 下面程序的时间复杂度为 $O(n)$。

```
i = 1
while i <= n:
    print(i)
    i *= 2
```

第 6 题 数组 `a` 已经升序排列，下面函数可以返回数组中最后一个小于等于 `x` 的元素下标；如果不存在该元素，则返回 `-1`。

```python
def findLastLE(a, x):
    l = 0
    r = len(a) - 1
    ans = -1
    while l <= r:
        mid = l + (r - l) // 2
        if a[mid] <= x:
            ans = mid
            l = mid + 1
        else:
            r = mid - 1
    return ans
```

第 7 题 快速排序中如果选取区间第一个元素作为枢轴。当输入数组已经升序排列时，其最坏时间复杂度仍为 $O(n \log n)$。

第 8 题 下面归并排序的递推式对应的时间复杂度为 $O(n \log n)$。

$$T(n) = 2T(n/2) + O(n)$$

第 9 题 下面的贪心代码，对于任意硬币面值集合 `coins`，都一定可以求出凑出 `money` 所需要的最少硬币数量。

```python
count = 0
# coins 按面值从大到小排序
for coin in coins:
    count += money // coin
    money = money % coin
```

第 10 题 假设两个非负高精度整数分别存储在数组 `a` 和 `b` 中，且 $a \geq b$。数组采用低位在前的方式存储，即 `a[0]` 表示个位。下面代码中的 `c` 可以正确保存 `a - b` 的各位数字。

```python
borrow = 0
for i in range(len(a)):
    t = a[i] - b[i] + borrow
    if t < 0:
        t += 10
        borrow = 1
    else:
        borrow = 0
    c[i] = t
```

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：哥德巴赫猜想
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

众所周知，哥德巴赫猜想是说，任何大于 2 的偶数都能写成两个质数（素数）之和。例如：

- $4 = 2 + 2$
- $6 = 3 + 3$
- $8 = 3 + 5$

- $10 = 3 + 7 = 5 + 5$

聪明的你肯定想知道，对于大于 2 的偶数 n，它有多少种写成两个质数之和的方法。例如 4，6 和 8 都只有一种方法，10 有两种方法。请你编写程序计算这个问题的答案。

在本题中，我们认为两种方案不同，当且仅当两种分解方案包含的素数互不相同；即 $10 = 3 + 7$ 和 $10 = 7 + 3$ 是同一种方案，不能重复计数。

3.1.2 输入格式

一行，一个大于 2 的偶数 n。

3.1.3 输出格式

一行，一个整数，表示将 n 写成两个质数之和的方法数。

3.1.4 样例

3.1.5 输入样例 1

```
4
```

3.1.6 输出样例 1

```
1
```

3.1.7 输入样例 2

```
10
```

3.1.8 输出样例 2

```
2
```

3.1.9 数据范围

对于 40% 的测试点，保证 $4 \leq n \leq 100$。

对于所有测试点，保证 $4 \leq n \leq 10^6$。

3.1.10 参考程序

```python
n = int(input())
is_primes = [True] * (n + 1)
is_primes[1] = False
# 埃氏筛
# 遍历每一个数
for i in range(2, n + 1):
    if is_primes[i]:
        # 对于一个素数，不断地去掉能被这个素数整除的数
        for j in range(2, n):
            if i * j > n:
                break
            is_primes[i * j] = False

ans = 0
# 遍历每一个可能性
for i in range(2, n // 2 + 1):
    if is_primes[i] and is_primes[n - i]:
        ans += 1
print(ans)
```

3.2 编程题 2

- **试题名称**：饮品调制
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

你想调制一份甜度恰到好处的饮品给你的朋友们品尝。

有 n 种原料可供用于调制饮品。第 i 种原料存量有 v_i 升，每升含有 s_i 克糖分。你可以自由选择原料加入饮品，但每种原料的使用量不得超过其剩余存量。也就是说，假设第 i 种原料选用 k_i 升，应当有 $0 \le k_i \le v_i$，k_i 可以取 0 到 v_i 之间的任何数字（包括小数）。

一份甜度恰到好处的饮品需要保证甜度恰好为 t。最终你调制得到的饮品甜度将为 $\frac{\sum_{i=1}^{n} k_i \cdot s_i}{\sum_{i=1}^{n} k_i}$。为了让更多的朋友喝到饮品，请问最多能调制出多少升甜度恰到好处的饮品？如果无法调制出甜度恰到好处的饮品，则认为答案是 0。

3.2.2 输入格式

第一行，两个整数 n, t，分别表示原料种类数量，恰到好处的甜度。

接下来 n 行，每行两个整数 v_i, s_i，分别表示第 i 种原料的存量体积，每升含有的糖分质量。

3.2.3 输出格式

一行，一个小数，表示能调制出的甜度恰到好处的饮品最大体积，保留三位小数。

3.2.4 样例

3.2.5 输入样例 1

```
4 2
6 1
5 2
8 5
1 0
```

3.2.6 输出样例 1

```
14.667
```

3.2.7 输入样例 2

```
2 5
3 4
5 3
```

3.2.8 输出样例 2

```
0.000
```

3.2.9 数据范围

对于 40% 的测试点，保证 $n = 2$。

对于所有测试点，保证 $1 \leq n \leq 2000$，$0 \leq t \leq 200$，$1 \leq v_i \leq 100$，$0 \leq s_i \leq 200$。

3.2.10 参考程序

```python
n, t = map(int, input().split())

def sugar(x):
    return x[1]
# 按照甜度，从低到高排序
e = sorted([list(map(int, input().split())) for _ in range(n)], key=sugar)

ans = 0
sv = ss = 0  # 当前总体积、总糖分
# 从低甜度到高甜度，尽量加入整份原料
for cv, cs in e:
    if ss + cs * cv > t * (sv + cv):
        # 加入整份后太甜，只加入 k 升，使总糖分等于 t 倍总体积
        k = (t * sv - ss) / (cs - t)
        ans = max(ans, sv + k)
        break
    ss += cs * cv
    sv += cv
else:
    # 所有原料恰好能全部使用
    if ss == t * sv:
        ans = max(ans, sv)

sv = ss = 0
# 从高甜度到低甜度，再扫描一次
for cv, cs in reversed(e):
    if ss + cs * cv < t * (sv + cv):
        # 加入整份后不够甜，只加入 k 升
        k = (t * sv - ss) / (cs - t)
        ans = max(ans, sv + k)
        break
    ss += cs * cv
    sv += cv
else:
    if ss == t * sv:
        ans = max(ans, sv)

print('%.3f' % ans)
```