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

Python 五级

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

第 1 题 以下哪种情况使用链表比数组更合适？

- A. 数据量固定且读多写少
- B. 需要频繁在中间或开头插入、删除元素
- C. 需要高效随机访问元素
- D. 存储空间必须连续

第 2 题 下面的python代码实现给定单链表头结点 `head` 和一个整数 `val`，删除链表中所有结点值等于 `val` 的节点，并返回新的头结点，则横线处填写（ ）。

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def removeElements(head: ListNode, val: int) -> ListNode:
    dummy = ListNode(0, head)
    cur = dummy
    while cur.next:
        if cur.next.val == val:
            ____________________________
            del del_node
        else:
            cur = cur.next
    return dummy.next
```

- A.

```python
del_node = del_node.
cur.next = del_node.next
```

- B.

```python
del_node = cur.next
cur = del_node.next
```

- C.

```python
del_node = cur.next
cur.next = del_node.next
```

- D.

```
del_node = cur.next
cur.next = del_node
```

第 3 题 下列python代码用Floyd判断一个单链表中是否存在环，链表的头节点为 `head` ，即用两个指针在链表上前进： `slow` 每次走 1 步， `fast` 每次走 2 步，若存在环， `fast` 终会追上 `slow` （相遇）；若无环， `fast` 会先到达 nullptr。横线上应填写（ ）。

```
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def hasCycle(head: ListNode) -> bool:
    if not head or not head.next:
        return False
    slow = head
    fast = head.next
    while fast and fast.next:
        if slow == fast:
            return True
        _________________
    return False

```

- [ ] **A.**

```
  slow = slow.next
  fast = fast.next.next
```

- [ ] **B.**

```
  slow.next = slow
  fast = fast.next.next
```

- [ ] **C.**

```
  slow = slow.next
  fast.next = fast.next.next
```

- [ ] **D.**

```
  slow = slow.next
  fast = fast.next
```

第 4 题 4.下列代码用于判断一个数是否为完全数(即等于它的真因子之和的数，如6=1+2+3)，哪个选项是正确的实现?

```
def isPerfectNumber(n: int) -> bool:
    if n <= 1:
        return False
    sum = 1
    i = 2
    while i * i <= n:
        if n % i == 0:
            sum += i

            ---------------
                sum += n // i
        i += 1
    return sum == n
```

- [ ] **A.** `if i != n / i:`
- [ ] **B.** `if i != n // i:`

- [ ] **C.** `if i = n // i:`
- [ ] **D.** `if i == n // i:`

第 5 题 以下代码计算两个数的最大公约数(GCD)，横线上应填写（ ）。

```
def gcd(a: int, b: int) -> int:
    if a < b:
        a, b = b, a  # 交换a和b的值
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return _________
```

- [ ] **A.** `b`
- [ ] **B.** `a`
- [ ] **C.** `temp`
- [ ] **D.** `a+b`

第 6 题 下面的代码实现埃拉托斯特尼筛法(埃氏筛)，横线处应填入（ ）。

```
def sieve(n: int):
    is_prime = [True] * (n + 1)
    is_prime[0] = is_prime[1] = False
    for i in range(2, n + 1):
        if is_prime[i]:
            for j in range(______, n + 1, i):
                is_prime[j] = False
    return is_prime

```

- [ ] **A.** `i`
- [ ] **B.** `i+1`
- [ ] **C.** `i*2`
- [ ] **D.** `i*i`

第 7 题 下面的代码实现线性筛法(欧拉筛)，横线处应填入（ ）。

```
def linearSieve(n: int):
    is_prime = [True] * (n + 1)
    primes = []
    for i in range(2, n + 1):
        if is_prime[i]:
            primes.append(i)
        for p in primes:
            if p * i > n:
                break
            is_prime[p * i] = False
            if _____________:
                break
    return primes
```

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

第 8 题 线性筛算法中有语句 `if p * i > n break;`，其目的是（ ）。

```
def linearSieve(n: int):
    is_prime = [True] * (n + 1)
    primes = []
    for i in range(2, n + 1):
        if is_prime[i]:
            primes.append(i)
        for p in primes:
            if p * i > n:
                break
            is_prime[p * i] = False
            if _____________:
                break
    return primes
```

- [ ] **A.** 降低常数但复杂度仍是 $O(n \log n)$
- [ ] **B.** 保证每个合数只被其最小质因子筛到一次，从而 $O(n)$
- [ ] **C.** 提高缓存命中率，复杂度仍 $O(n \log \log n)$
- [ ] **D.** 不重要，是否 `break` 都一样

第 9 题 唯一分解定理描述的是( )。

- [ ] **A.** 每个整数都能表示为任意素数的乘积
- [ ] **B.** 每个大于 1 的整数能唯一分解为素数幂乘积（忽略顺序）
- [ ] **C.** 合数不能分解为素数乘积
- [ ] **D.** 素数只有两个因子：1 和自身

第 10 题 给定一个 `n x n` 的矩阵 `matrix`，矩阵的每一行和每一列都按升序排列。下面代码返回矩阵中第 `k` 小的元素，则两处横线上应分别填写（ ）。

```
def countLE(matrix, x):
    n = len(matrix)
    i, j = n - 1, 0
    cnt = 0
    while i >= 0 and j < n:
        if matrix[i][j] <= x:
            cnt += i + 1
            j += 1
        else:
            i -= 1
    return cnt

def kthSmallest(matrix, k):
    n = len(matrix)
    lo = matrix[0][0]
    hi = matrix[n-1][n-1]

    while lo < hi:
        mid = lo + (hi - lo) // 2
        if countLE(matrix, mid) >= k:
            ________
        else:
            ________
    return lo
```

- [ ] **A.**

```
hi = mid - 1;
lo = mid + 1;
```

☐ **B.**

```
hi = mid;
lo = mid;
```

☐ **C.**

```
hi = mid;
lo = mid + 1;
```

☐ **D.**

```
hi = mid + 1;
lo = mid;
```

第 11 题 下述python代码实现了快速排序算法，下面说法错误的是（ ）。

```
def partition(arr, low, high):

    i, j = low, high
    while i < j:
        while i < j and arr[j] >= arr[low]:
            j -= 1
        while i < j and arr[i] <= arr[low]:
            i += 1
        arr[i], arr[j] = arr[j], arr[i]
    arr[i], arr[low] = arr[low], arr[i]
    return i

def quickSort(arr, low, high):
    if low < high:
        pivot = partition(arr, low, high)
        quickSort(arr, low, pivot - 1)
        quickSort(arr, pivot + 1, high)
```

☐ **A.** 快速排序平均情况下比归并排序的比较、赋值、交换等操作的总数量少，所以命名为“快速”。

☐ **B.** 在平均情况下，划分的递归层数为 $\log n$，每层中的总循环数为n ，总时间为 $O(n \log n)$。

☐ **C.** 在最差情况下，每轮划分操作都将长度为n的数组划分为长度为 0 和 $n-1$ 的两个子数组，此时递归层数达到 n ，每层中的循环数为n，总时间为 $O\left(n^2\right)$。

☐ **D.** 划分函数中“从右往左查找”与“从左往右查找”的顺序可以互换。

第 12 题 下述python代码实现了归并排序算法，则横线上应填写（ ）。

```
def merge(nums, left, mid, right):
    tmp = []
    i, j = left, mid + 1
    while i <= mid and j <= right:
        if nums[i] <= nums[j]:
            tmp.append(nums[i])
            i += 1
        else:
            tmp.append(nums[j])
            j += 1

    while i <= mid:
        tmp.append(nums[i])
        i += 1

    while ___________:
```

```
        tmp.append(nums[j])
        j += 1

    for k in range(len(tmp)):
        nums[left + k] = tmp[k]

def mergeSort(nums, left, right):
    if left >= right:
        return

    mid = (left + right) // 2
    mergeSort(nums, left, mid)
    mergeSort(nums, mid + 1, right)
    merge(nums, left, mid, right)

# 使用示例
if __name__ == "__main__":
    nums = [3, 1, 4, 1, 5, 9, 2, 6]
    mergeSort(nums, 0, len(nums) - 1)
    print(nums)  # 输出排序后的数组

```

- [ ] **A.** `i < mid`
- [ ] **B.** `j < right`
- [ ] **C.** `i <= mid`
- [ ] **D.** `j <= right`

第 13 题 假设你是一家电影院的排片经理，只有一个放映厅。你有一个电影列表 `movies`，其中 `movies[i] = [start_i, end_i]` 表示第 `i` 部电影的开始和结束时间。请你找出最多能安排多少部不重叠的电影，则横线上应分别填写的代码为（ ）。

```
def maxMovies(movies):
    if not movies:
        return 0

    # 按照结束时间排序
    movies.sort(key=lambda x: x[1])

    count = 1
    ____________ = movies[0][1]

    for i in range(1, len(movies)):
        if movies[i][0] >= lastEnd:
            count += 1
            lastEnd = movies[i][1]

    return count
```

- [ ] **A.** `movies`
- [ ] **B.** `lastEnd`
- [ ] **C.** `movies[i][0]`
- [ ] **D.** `movies[i][1]`

第 14 题 给定一个整数数组 `nums`，下面代码找到一个具有最大和的连续子数组，并返回其最大和。则下面说法错误的是（ ）。

```
import math

def crossSum(nums, left, mid, right):
```

```
    leftSum = -math.inf
    sum_val = 0
    for i in range(mid, left - 1, -1):
        sum_val += nums[i]
        leftSum = max(leftSum, sum_val)

    rightSum = -math.inf
    sum_val = 0
    for i in range(mid + 1, right + 1):
        sum_val += nums[i]
        rightSum = max(rightSum, sum_val)

    return leftSum + rightSum

def helper(nums, left, right):
    if left == right:
        return nums[left]

    mid = left + (right - left) // 2
    leftMax = helper(nums, left, mid)
    rightMax = helper(nums, mid + 1, right)
    crossMax = crossSum(nums, left, mid, right)

    return max(leftMax, rightMax, crossMax)

def maxSubArray(nums):
    return helper(nums, 0, len(nums) - 1)
```

- [ ] **A.** 上述代码采用分治算法实现
- [ ] **B.** 本题算法采用贪心算法
- [ ] **C.** 时间复杂度：$O(\log n)$
- [ ] **D.** 由于采用递归方式实现，空间复杂度：$O(\log n)$

第 15 题 给定一个由非负整数组成的数组 `digits`，表示一个非负整数的各位数字（最高位在数组首位）。下面代码对该整数执行 +1 操作，并返回结果数组，则横线上应填写（ ）。

```
def plusOne(digits):
    for i in range(len(digits)-1, -1, -1):
        if digits[i] < 9:
            digits[i] += 1
            return digits
        digits[i] = 0

    ---------------
```

- [ ] **A.**

```
    digits.insert(0, 1)
    return digits
```

- [ ] **B.**

```
    digits.insert(1, 0)
    return digits
```

- [ ] **C.**

```
    digits.insert(0, 1)
```

- [ ] **D.**

```
    digits.insert(1, 1)
    return digits
```

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 基于下面定义的函数，通过判断 `isDivisibleBy9(n) == isDigitSumDivisibleBy9(n)` 代码可验算如果一个数能被9整除，则它的各位数字之和能被9整除。

```
def isDivisibleBy9(n):
    return n % 9 == 0

def isDigitSumDivisibleBy9(n):
    num_str = str(n)
    digit_sum = 0
    for c in num_str:
        digit_sum += int(c)
    return digit_sum % 9 == 0
```

第 2 题 假设函数 `gcd()` 函数能正确求两个正整数的最大公约数，则下面的 `findMusicalPattern(4, 6)` 函数返回 2。

```
import math

def findMusicalPattern(rhythm1, rhythm2):
    commonDivisor = math.gcd(rhythm1, rhythm2)
    patternLength = (rhythm1 * rhythm2) // commonDivisor
```

第 3 题 下面递归实现的斐波那契数列的时间复杂度为 $O(2^n)$。

```
def fib_memo(n, memo):
    if n <= 1:
        return n
    if memo[n] != -1:
        return memo[n]
    memo[n] = fib_memo(n - 1, memo) + fib_memo(n - 2, memo)
    return memo[n]

if __name__ == "__main__":
    n = 40
    memo = [-1] * 100
    result = fib_memo(n, memo)
    print(result)
```

第 4 题 链表通过更改指针实现高效的节点插入与删除，但节点访问效率低、占用内存较多，且对缓存利用不友好。

第 5 题 二分查找依赖数据的有序性，通过循环逐步缩减一半搜索区间来进行查找，且仅适用于数组或基于数组实现的数据结构。

第 6 题 线性筛关键是“每个合数只会被最小质因子筛到一次”，因此为 $O(n)$。

第 7 题 快速排序和归并排序都是稳定的排序算法。

第 8 题 下面代码采用分治算法求解汉诺塔问题，时间复杂度为 $O(n \log n)$。

```
def move(src, tar):
    pan = src.pop()
```

```
    tar.append(pan)

def dfs(n, src, buf, tar):
    if n == 1:
        move(src, tar)
        return

    dfs(n - 1, src, tar, buf)
    move(src, tar)
    dfs(n - 1, buf, src, tar)

def solveHanota(A, B, C):
    n = len(A)
    dfs(n, A, B, C)
```

第 9 题 所有递归算法都可以转换为迭代算法。

第 10 题 贪心算法总能得到全局最优解。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：数字选取
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

给定正整数 n，现在有 $1, 2, \ldots, n$ 共计 n 个整数。你需要从这 n 个整数中选取一些整数，使得所选取的整数中任意两个不同的整数均互质（也就是说，这两个整数的最大公因数为 1）。请你最大化所选取整数的数量。

例如，当 $n = 9$ 时，可以选择 $1, 5, 7, 8, 9$ 共计 5 个整数。可以验证不存在数量更多的选取整数的方案。

3.1.2 输入格式

一行，一个正整数 n，表示给定的正整数。

3.1.3 输出格式

一行，一个正整数，表示所选取整数的最大数量。

3.1.4 样例

3.1.4.1 输入样例 1

```
6
```

3.1.4.2 输出样例 1

```
4
```

3.1.4.3 输入样例 2

```
9
```

3.1.4.4 输出样例 2

```
5
```

3.1.5 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 1000$。

对于所有测试点，保证 $1 \leq n \leq 10^5$。

3.1.6 参考程序

```python
n = int(input())  # 输入n
p = [0] + [1] * n  # 各个数字是否被选中，p[i]=1表示被选中
for i in range(2, int(n**0.5) + 1):  # 假设一个数小于等于n，且不能被int(n**0.5)以下的数整除，那么这个数必定为质数
    if not p[i]:
        continue
    for j in range(i, n + 1):
        if i * j > n:
            break
        p[i * j] = 0  # 排除掉能够整除i的数
print(sum(p))
```

3.2 编程题 2

- **试题名称**：有趣的数字和
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

如果一个正整数的二进制表示包含奇数个 1，那么小 A 就会认为这个正整数是**有趣的**。

例如，7 的二进制表示为 $(111)_2$，包含 1 的个数为 3 个，所以 7 是有趣的。但是 $9 = (1001)_2$ 包含 2 个 1，所以 9 不是有趣的。

给定正整数 l, r，请你统计满足 $l \leq n \leq r$ 的有趣的整数 n 之和。

3.2.2 输入格式

一行，两个正整数 l, r，表示给定的正整数。

3.2.3 输出格式

一行，一个正整数，表示 l, r 之间有趣的整数之和。

3.2.4 样例

3.2.4.1 输入样例 1

```
3 8
```

3.2.4.2 输出样例 1

```
19
```

3.2.4.3 输入样例 2

```
65 36248
```

3.2.4.4 输出样例 2

```
328505490
```

3.2.5 数据范围

对于 40% 的测试点，保证 $1 \leq l \leq r \leq 10^4$。

对于另外 30% 的测试点，保证 $l = 1$ 并且 $r = 2^k - 1$，其中 k 是大于 1 的正整数。

对于所有测试点，保证 $1 \leq l \leq r \leq 10^9$。

3.2.6 参考程序

```python
def cal2(n, p):
    """
    计算奇偶性为p的，0~n的有趣的数之和，但是要求n为2^k-1的形式
    :return 有趣的数数量，有趣的数之和
    """
    if n == 0:
        return 1 - p, 0
    if n == 1:
        return 1, p
    return (n + 1) // 2, n * (n + 1) // 4

def cal(n, p):
    """
    计算奇偶性为p的0~n的有趣的数之和
    :return 有趣的数数量，有趣的数之和
    """
    if n <= 1:
        return cal2(n, p)
    x = 1
    while x * 2 <= n:
        x *= 2
    # 找出一个k，是的2^k-1<=n，并且k尽可能大，使用上面的简单情况计算
    cntl, sl = cal2(x - 1, p)
    # 剩下的部分，减去2^k之后，相当于最高位的1被拿掉了，所以有趣的数剩下的各个数位之和要和p相反
    cntr, sr = cal(n - x, 1 - p)
    # 有趣的数数量是两部分之和
    # 有趣的数之和左边就是cal2的计算结果，右边就是cal的计算结果，再乘以有趣的数被去掉的2^k
    return cntl + cntr, sl + sr + x * cntr

l, r = map(int, input().split())
# 分别计算0~l-1和0~r的有趣的数之和，再做差
_, ansl = cal(l - 1, 1)
_, ansr = cal(r, 1)
print(ansr - ansl)
```