200 Python Coding Interview Questions (With Code Solutions)

Part 1: Basic Python & Syntax (Questions 1-20)

1. Write a program to check if a number is even or odd.

Python
```
n = int(input("Enter number: "))
print("Even" if n % 2 == 0 else "Odd")
```

2. Swap two variables without a third variable.

Python
```
a, b = 5, 10
a, b = b, a
print(a, b)  # Output: 10, 5
```

3. Check if a number is prime.

Python
```
n = 11
if n > 1:
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            print("Not Prime")
            break
    else: print("Prime")
else: print("Not Prime")
```

4. Factorial using recursion.

Python
```
def factorial(n):
    return 1 if n == 0 else n * factorial(n - 1)
print(factorial(5))
```

5. Generate Fibonacci sequence up to N terms.

```
Python
n = 10
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

6. Check if a year is a leap year.

```
Python
y = 2024
if (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0):
    print("Leap")
else:
    print("Not Leap")
```

7. Print Armstrong numbers between 100 and 1000.

```
Python
for num in range(100, 1000):
    if num == sum(int(d)**3 for d in str(num)):
        print(num)
```

8. Reverse an integer.

```
Python
n = 12345
print(int(str(n)[::-1]))
```

9. Sum of digits of a number.

```
Python
n = 123
print(sum(int(d) for d in str(n)))
```

10. Find GCD of two numbers.

```
Python
import math
```

```
print(math.gcd(48, 18))
```

11. Sum of numbers 1 to 100 (one-liner).

```
Python
print(sum(range(1, 101)))
```

12. Create an infinite loop.

```
Python
while True: pass
```

13. Difference between `break` and `continue`? *Code:* `break` exits the loop; `continue` skips to the next iteration.

14. Convert Celsius to Fahrenheit.

```
Python
c = 25
f = (c * 9/5) + 32
print(f)
```

15. Take multiple inputs in one line.

```
Python
x, y = map(int, input().split())
```

16. What does `pass` do? *Answer:* Null statement; does nothing. Used as a placeholder.

17. Check data type of a variable.

```
Python
x = 5
print(type(x))
```

18. Find largest of three numbers.

```
Python
print(max(10, 20, 15))
```

19. Generate random number between 1 and 100.

```
Python
import random
print(random.randint(1, 100))
```

20. Ternary operator example.

```
Python
res = "Pass" if 80 > 40 else "Fail"
```

Part 2: String Manipulation (Questions 21-40)

21. Reverse a string.

```
Python
s = "Python"
print(s[::-1])
```

22. Check if palindrome.

```
Python
s = "madam"
print(s == s[::-1])
```

23. Count character occurrences.

```
Python
from collections import Counter
print(Counter("hello"))
```

24. Check if anagrams.

```
Python
print(sorted("listen") == sorted("silent"))
```

25. Remove duplicates from string.

```
Python
```

```
s = "hello"
print("".join(sorted(set(s), key=s.index)))
```

26. First non-repeating character.

Python
```
s = "swiss"
for c in s:
    if s.count(c) == 1:
        print(c); break
```

27. String to list of words.

Python
```
print("Python is great".split())
```

28. List of words to string.

Python
```
print(" ".join(['Python', 'is', 'great']))
```

29. Check if string is numeric.

Python
```
print("123".isdigit())
```

30. Replace character.

Python
```
print("hello".replace('l', 'z'))
```

31. Count vowels and consonants.

Python
```
s = "hello"
v = sum(1 for c in s if c in 'aeiou')
c = sum(1 for c in s if c.isalpha() and c not in 'aeiou')
print(v, c)
```

32. Longest word in sentence.

```
Python
s = "Python programming"
print(max(s.split(), key=len))
```

33. Toggle case.

```
Python
print("PyThOn".swapcase())
```

34. Check start of string.

```
Python
print("Hello".startswith("He"))
```

35. Get all substrings.

```
Python
s = "abc"
print([s[i:j] for i in range(len(s)) for j in range(i+1, len(s)+1)])
```

36. Remove whitespace.

```
Python
print("  hi  ".strip())
```

37. Convert to Title Case.

```
Python
print("hello world".title())
```

38. Check valid identifier.

```
Python
print("var_1".isidentifier())
```

39. Count words in file.

```
Python
```

```
# print(len(open("file.txt").read().split()))
```

40. F-string formatting.

```
Python
name = "Ali"
print(f"Hello {name}")
```

Part 3: Lists & Arrays (Questions 41-60)

41. Max and Min in list.

```
Python
l = [1, 5, 2]
print(max(l), min(l))
```

42. Second largest number.

```
Python
l = [10, 20, 4, 45, 99]
print(sorted(list(set(l)))[-2])
```

43. Remove duplicates from list.

```
Python
l = [1, 2, 2, 3]
print(list(set(l)))
```

44. Sort list without `sort()`.

```
Python
l = [3, 1, 2]
for i in range(len(l)):
    for j in range(len(l)-1):
        if l[j] > l[j+1]: l[j], l[j+1] = l[j+1], l[j]
print(l)
```

45. Reverse a list.

```
Python
l = [1, 2, 3]
l.reverse()
print(l)
```

46. Merge sorted lists.

```
Python
l1, l2 = [1, 3], [2, 4]
print(sorted(l1 + l2))
```

47. Common elements.

```
Python
print(list(set([1,2]) & set([2,3])))
```

48. Squares list comprehension.

```
Python
print([x**2 for x in range(5)])
```

49. Flatten nested list.

```
Python
l = [[1,2], [3,4]]
print([i for sub in l for i in sub])
```

50. Shuffle list.

```
Python
import random
l = [1, 2, 3]
random.shuffle(l)
print(l)
```

51. Sum of elements.

```
Python
print(sum([1, 2, 3]))
```

52. Check empty list.

```
Python
l = []
if not l: print("Empty")
```

53. Find index of element.

```
Python
print([10, 20].index(20))
```

54. Count frequency.

```
Python
print([1, 2, 2].count(2))
```

55. Split list into chunks.

```
Python
l = [1, 2, 3, 4]
n = 2
print([l[i:i+n] for i in range(0, len(l), n)])
```

56. `append` vs `extend`. *Code:* `[1].append([2])` -> `[1, [2]]`. `[1].extend([2])` -> `[1, 2]`.

57. Remove Nth element.

```
Python
l = [10, 20, 30]
l.pop(1)
print(l)
```

58. Rotate list by K.

```
Python
l = [1, 2, 3, 4, 5]
k = 2
print(l[-k:] + l[:-k])
```

59. Check list equality.

```
Python
print([1, 2] == [1, 2])
```

60. Sieve of Eratosthenes (Primes).

```
Python
def sieve(n):
    primes = [True]*(n+1)
    for p in range(2, int(n**0.5)+1):
        if primes[p]:
            for i in range(p*p, n+1, p): primes[i] = False
    return [p for p in range(2, n) if primes[p]]
```

Part 4: Dictionaries & Sets (Questions 61-75)

61. Sort dict by value.

```
Python
d = {'a': 2, 'b': 1}
print(dict(sorted(d.items(), key=lambda x: x[1])))
```

62. Merge two dicts.

```
Python
d1, d2 = {'a': 1}, {'b': 2}
print(d1 | d2)
```

63. Iterate dict.

```
Python
for k, v in {'a':1}.items(): print(k, v)
```

64. Check key exists.

```
Python
print('a' in {'a': 1})
```

65. Remove key.

```
Python
d = {'a': 1}; d.pop('a')
```

66. defaultdict example.

```
Python
from collections import defaultdict
d = defaultdict(int); d['a'] += 1
```

67. Dict from two lists.

```
Python
print(dict(zip(['a', 'b'], [1, 2])))
```

68. Set intersection.

```
Python
print({1,2} & {2,3})
```

69. Remove duplicates via set.

```
Python
print(list(set([1, 1, 2])))
```

70. Word frequency dict.

```
Python
text = "hi hi world"
freq = {}
for w in text.split(): freq[w] = freq.get(w, 0) + 1
print(freq)
```

71. Set vs Dict. *Answer:* Set = unique keys. Dict = key-value pairs.

72. Get dict keys.

```
Python
print(list({'a':1}.keys()))
```

73. Dict comprehension.

```Python
print({x: x**2 for x in range(3)})
```

74. Clear set.

```Python
s = {1}; s.clear()
```

75. Set difference.

```Python
print({1, 2} - {1})
```

Part 5: OOP & Advanced (Questions 76-90)

76. Define a class.

```Python
class Dog:
    def bark(self): print("Woof")
```

77. Inheritance.

```Python
class A: pass
class B(A): pass
```

78. `__init__`. *Answer:* Constructor method.

79. `self`. *Answer:* Reference to current instance.

80. Class vs Instance var.

```Python
class A:
    c_var = 1
    def __init__(self): self.i_var = 2
```

81. Private variable.

```
Python
class A: __private = 10
```

82. Decorator.

```
Python
def dec(func):
    def wrapper(): print("Pre"); func()
    return wrapper
```

83. Generator.

```
Python
def gen(): yield 1
```

84. Lambda.

```
Python
add = lambda x,y: x+y
```

85. Exception handling.

```
Python
try: 1/0
except: print("Error")
finally: print("Done")
```

86. Global keyword.

```
Python
x = 10
def change(): global x; x=20
```

87. Read file line by line.

```
Python
# with open('f.txt') as f:
```

```
#     for l in f: print(l)
```

88. List Comprehension. *Answer:* Concise list creation `[x for x in iter]`.

89. `*args`, `**kwargs`. *Answer:* Variable positional and keyword arguments.

90. Deep vs Shallow copy.

```
Python
import copy
copy.deepcopy([])
```

Part 6: Algorithms/Misc (Questions 91-100)

91. Binary Search.

```
Python
def binary_search(arr, x):
    l, r = 0, len(arr)-1
    while l <= r:
        mid = (l+r)//2
        if arr[mid] == x: return mid
        elif arr[mid] < x: l = mid+1
        else: r = mid-1
    return -1
```

92. Map two lists to dict.

```
Python
print(dict(zip(['a'], [1])))
```

93. Print Pyramid.

```
Python
n=5
for i in range(n): print(" "*(n-i-1) + "*"*(2*i+1))
```

94. Missing number 1 to N.

```
Python
def missing(arr, n): return n*(n+1)//2 - sum(arr)
```

95. `with` statement. *Answer:* Context manager (auto-close).

96. String to datetime.

```
Python
from datetime import datetime
print(datetime.strptime("2024-01-01", "%Y-%m-%d"))
```

97. Transpose Matrix.

```
Python
m = [[1,2],[3,4]]
print([[r[i] for r in m] for i in range(2)])
```

98. Get CWD.

```
Python
import os; print(os.getcwd())
```

99. Balanced Parentheses.

```
Python
def is_balanced(s):
    stack = []
    pairs = {')':'(', '}':'{', ']':'['}
    for c in s:
        if c in pairs.values(): stack.append(c)
        elif c in pairs and (not stack or stack.pop() != pairs[c]): return False
    return not stack
```

100. Measure execution time.

```
Python
import time
s = time.time(); time.sleep(1); print(time.time()-s)
```

Part 7: Advanced Arrays (Questions 101-120)

101. Two Sum.

```
Python
def twoSum(nums, target):
    seen = {}
    for i, n in enumerate(nums):
        if target - n in seen: return [seen[target-n], i]
        seen[n] = i
```

102. Best Time to Buy and Sell Stock.

```
Python
def maxProfit(prices):
    minP, maxP = float('inf'), 0
    for p in prices:
        minP = min(minP, p)
        maxP = max(maxP, p - minP)
    return maxP
```

103. Product Except Self.

```
Python
def productExceptSelf(nums):
    res = [1] * len(nums)
    prefix = 1
    for i in range(len(nums)):
        res[i] = prefix; prefix *= nums[i]
    postfix = 1
    for i in range(len(nums)-1, -1, -1):
        res[i] *= postfix; postfix *= nums[i]
    return res
```

104. Container With Most Water.

```
Python
def maxArea(height):
    l, r, res = 0, len(height)-1, 0
    while l < r:
        res = max(res, (r-l) * min(height[l], height[r]))
        if height[l] < height[r]: l += 1
        else: r -= 1
    return res
```

105. 3Sum.

```
Python
def threeSum(nums):
    nums.sort(); res = []
    for i, a in enumerate(nums):
        if i > 0 and a == nums[i-1]: continue
        l, r = i+1, len(nums)-1
        while l < r:
            s = a + nums[l] + nums[r]
            if s > 0: r -= 1
            elif s < 0: l += 1
            else:
                res.append([a, nums[l], nums[r]])
                l+=1
                while nums[l]==nums[l-1] and l<r: l+=1
    return res
```

106. Maximum Subarray (Kadane).

```
Python
def maxSubArray(nums):
    curSum, maxSum = 0, nums[0]
    for n in nums:
        curSum = max(curSum + n, n)
        maxSum = max(maxSum, curSum)
    return maxSum
```

107. Rotate Image.

```
Python
def rotate(matrix):
    matrix[:] = matrix[::-1]
    for i in range(len(matrix)):
        for j in range(i):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
```

108. Spiral Matrix.

```python
def spiralOrder(matrix):
    return matrix and [*matrix.pop(0)] + spiralOrder([*zip(*matrix)][::-1])
```

109. Search Rotated Sorted Array.

```python
def search(nums, target):
    l, r = 0, len(nums)-1
    while l <= r:
        mid = (l+r)//2
        if nums[mid] == target: return mid
        if nums[l] <= nums[mid]:
            if nums[l] <= target < nums[mid]: r = mid-1
            else: l = mid+1
        else:
            if nums[mid] < target <= nums[r]: l = mid+1
            else: r = mid-1
    return -1
```

110. Merge Intervals.

```python
def merge(intervals):
    intervals.sort()
    res = [intervals[0]]
    for start, end in intervals[1:]:
        if start <= res[-1][1]: res[-1][1] = max(res[-1][1], end)
        else: res.append([start, end])
    return res
```

111. Insert Interval.

```python
def insert(intervals, newInterval):
    res = []
    for i, interval in enumerate(intervals):
        if newInterval[1] < interval[0]:
            res.append(newInterval); return res + intervals[i:]
        elif newInterval[0] > interval[1]:
            res.append(interval)
        else:
            newInterval = [min(newInterval[0], interval[0]), max(newInterval[1], interval[1])]
    res.append(newInterval)
    return res
```

112. Longest Consecutive Sequence.

```
Python
def longestConsecutive(nums):
    nums = set(nums); res = 0
    for n in nums:
        if n-1 not in nums:
            l = 0
            while n+l in nums: l += 1
            res = max(res, l)
    return res
```

113. Move Zeroes.

```
Python
def moveZeroes(nums):
    l = 0
    for r in range(len(nums)):
        if nums[r]:
            nums[l], nums[r] = nums[r], nums[l]
            l += 1
```

114. Find Duplicate Number (Floyd's).

```
Python
def findDuplicate(nums):
    s, f = 0, 0
    while True:
        s, f = nums[s], nums[nums[f]]
        if s == f: break
    s2 = 0
    while True:
        s, s2 = nums[s], nums[s2]
        if s == s2: return s
```

115. Subarray Sum Equals K.

```
Python
```

```
def subarraySum(nums, k):
    res = cur = 0
    prefix = {0: 1}
    for n in nums:
        cur += n
        res += prefix.get(cur - k, 0)
        prefix[cur] = prefix.get(cur, 0) + 1
    return res
```

116. Sliding Window Maximum.

```
Python
from collections import deque
def maxSlidingWindow(nums, k):
    q, res = deque(), []
    for i, n in enumerate(nums):
        while q and nums[q[-1]] < n: q.pop()
        q.append(i)
        if q[0] == i - k: q.popleft()
        if i >= k - 1: res.append(nums[q[0]])
    return res
```

117. Trapping Rain Water.

```
Python
def trap(height):
    l, r = 0, len(height)-1
    leftMax, rightMax = height[l], height[r]
    res = 0
    while l < r:
        if leftMax < rightMax:
            l += 1; leftMax = max(leftMax, height[l])
            res += leftMax - height[l]
        else:
            r -= 1; rightMax = max(rightMax, height[r])
            res += rightMax - height[r]
    return res
```

118. First Missing Positive.

```
Python
def firstMissingPositive(nums):
    n = len(nums)
    for i in range(n):
        while 1 <= nums[i] <= n and nums[nums[i]-1] != nums[i]:
            nums[nums[i]-1], nums[i] = nums[i], nums[nums[i]-1]
    for i in range(n):
        if nums[i] != i + 1: return i + 1
    return n + 1
```

119. Sort Colors.

```
Python
def sortColors(nums):
    l, r, i = 0, len(nums)-1, 0
    while i <= r:
        if nums[i] == 0: nums[l], nums[i] = nums[i], nums[l]; l+=1
        elif nums[i] == 2: nums[r], nums[i] = nums[i], nums[r]; r-=1; i-=1
        i += 1
```

120. Median of Two Sorted Arrays.

```
Python
def findMedianSortedArrays(nums1, nums2):
    A, B = (nums1, nums2) if len(nums1) < len(nums2) else (nums2, nums1)
    total = len(nums1) + len(nums2)
    half = total // 2
    l, r = 0, len(A) - 1
    while True:
        i = (l + r) // 2
        j = half - i - 2
        Aleft = A[i] if i >= 0 else float("-inf")
        Aright = A[i+1] if (i+1) < len(A) else float("inf")
        Bleft = B[j] if j >= 0 else float("-inf")
        Bright = B[j+1] if (j+1) < len(B) else float("inf")
        if Aleft <= Bright and Bleft <= Aright:
            if total % 2: return min(Aright, Bright)
            return (max(Aleft, Bleft) + min(Aright, Bright)) / 2
        elif Aleft > Bright: r = i - 1
        else: l = i + 1
```

Part 8: Linked Lists (Questions 121-135)

121. Reverse Linked List.

```
Python
def reverseList(head):
    prev, curr = None, head
    while curr:
        nxt = curr.next; curr.next = prev; prev = curr; curr = nxt
    return prev
```

122. Merge Two Sorted Lists.

```
Python
def mergeTwoLists(l1, l2):
    dummy = cur = ListNode()
    while l1 and l2:
        if l1.val < l2.val: cur.next = l1; l1 = l1.next
        else: cur.next = l2; l2 = l2.next
        cur = cur.next
    cur.next = l1 or l2
    return dummy.next
```

123. Reorder List.

```
Python
def reorderList(head):
    # Find middle
    slow, fast = head, head.next
    while fast and fast.next: slow = slow.next; fast = fast.next.next
    # Reverse second half
    second = slow.next; slow.next = None; prev = None
    while second: tmp = second.next; second.next = prev; prev = second; second = tmp
    # Merge
    first, second = head, prev
    while second:
        tmp1, tmp2 = first.next, second.next
        first.next = second; second.next = tmp1
        first, second = tmp1, tmp2
```

124. Remove Nth Node From End.

```
Python
def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    left, right = dummy, head
    for _ in range(n): right = right.next
    while right: left = left.next; right = right.next
    left.next = left.next.next
    return dummy.next
```

125. Cycle Detection.

```
Python
def hasCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next; fast = fast.next.next
        if slow == fast: return True
    return False
```

126. Add Two Numbers.

```
Python
def addTwoNumbers(l1, l2):
    dummy = cur = ListNode(); carry = 0
    while l1 or l2 or carry:
        v1 = l1.val if l1 else 0
        v2 = l2.val if l2 else 0
        val = v1 + v2 + carry
        carry = val // 10
        cur.next = ListNode(val % 10); cur = cur.next
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None
    return dummy.next
```

127. Intersection of Two Lists.

```
Python
def getIntersectionNode(headA, headB):
    l1, l2 = headA, headB
    while l1 != l2:
        l1 = l1.next if l1 else headB
        l2 = l2.next if l2 else headA
    return l1
```

128. Palindrome Linked List.

```
Python
def isPalindrome(head):
    vals = []
    while head: vals.append(head.val); head = head.next
    return vals == vals[::-1]
```

129. Reverse Nodes in k-Group.

```
Python
def reverseKGroup(head, k):
    dummy = ListNode(0, head)
    groupPrev = dummy
    while True:
        kth = groupPrev
        for _ in range(k):
            kth = kth.next
            if not kth: return dummy.next
        groupNext = kth.next

        # Reverse group
        prev, curr = kth.next, groupPrev.next
        while curr != groupNext:
            tmp = curr.next; curr.next = prev; prev = curr; curr = tmp

        tmp = groupPrev.next
        groupPrev.next = kth
        groupPrev = tmp
```

130. Copy List with Random Pointer.

```
Python
def copyRandomList(head):
    oldToNew = {None: None}
    cur = head
    while cur:
        oldToNew[cur] = Node(cur.val)
        cur = cur.next
    cur = head
    while cur:
        oldToNew[cur].next = oldToNew[cur.next]
        oldToNew[cur].random = oldToNew[cur.random]
        cur = cur.next
    return oldToNew[head]
```

131. Merge k Sorted Lists.

```
Python
import heapq
def mergeKLists(lists):
    h = []
    for i, l in enumerate(lists):
        if l: heapq.heappush(h, (l.val, i, l))
    dummy = cur = ListNode()
    while h:
        val, i, node = heapq.heappop(h)
        cur.next = node; cur = cur.next
        if node.next: heapq.heappush(h, (node.next.val, i, node.next))
    return dummy.next
```

132. Swap Nodes in Pairs.

```
Python
def swapPairs(head):
    dummy = ListNode(0, head)
    prev, curr = dummy, head
    while curr and curr.next:
        nxtPair = curr.next.next
        second = curr.next
        second.next = curr
        curr.next = nxtPair
        prev.next = second
        prev = curr
        curr = nxtPair
    return dummy.next
```

133. Sort List.

```
Python
def sortList(head):
    if not head or not head.next: return head
    # Split
    slow, fast = head, head.next
    while fast and fast.next: slow = slow.next; fast = fast.next.next
```

```python
    mid = slow.next; slow.next = None
    # Recurse
    left, right = sortList(head), sortList(mid)
    # Merge
    dummy = tail = ListNode()
    while left and right:
        if left.val < right.val: tail.next = left; left = left.next
        else: tail.next = right; right = right.next
        tail = tail.next
    tail.next = left or right
    return dummy.next
```

134. Rotate List.

Python
```python
def rotateRight(head, k):
    if not head: return head
    length, tail = 1, head
    while tail.next: tail = tail.next; length += 1
    k = k % length
    if k == 0: return head
    cur = head
    for _ in range(length - k - 1): cur = cur.next
    newHead = cur.next
    cur.next = None
    tail.next = head
    return newHead
```

135. LRU Cache.

Python
```python
class LRUCache:
    def __init__(self, capacity):
        self.cap = capacity; self.cache = OrderedDict()
    def get(self, key):
        if key not in self.cache: return -1
        self.cache.move_to_end(key)
        return self.cache[key]
    def put(self, key, value):
        if key in self.cache: self.cache.move_to_end(key)
        self.cache[key] = value
        if len(self.cache) > self.cap: self.cache.popitem(last=False)
```

Part 9: Advanced Strings (Questions 136-150)

136. Longest Substring Without Repeating.

```
Python
def lengthOfLongestSubstring(s):
    charSet = set(); l = 0; res = 0
    for r in range(len(s)):
        while s[r] in charSet:
            charSet.remove(s[l]); l += 1
        charSet.add(s[r])
        res = max(res, r - l + 1)
    return res
```

137. Longest Palindromic Substring.

```
Python
def longestPalindrome(s):
    res = ""
    for i in range(len(s)):
        # Odd
        l, r = i, i
        while l >= 0 and r < len(s) and s[l] == s[r]:
            if (r - l + 1) > len(res): res = s[l:r+1]
            l -= 1; r += 1
        # Even
        l, r = i, i + 1
        while l >= 0 and r < len(s) and s[l] == s[r]:
            if (r - l + 1) > len(res): res = s[l:r+1]
            l -= 1; r += 1
    return res
```

138. Group Anagrams.

```
Python
def groupAnagrams(strs):
    ans = defaultdict(list)
    for s in strs:
        key = [0]*26
        for c in s: key[ord(c)-ord('a')] += 1
        ans[tuple(key)].append(s)
    return list(ans.values())
```

139. Valid Parentheses.

```
Python
def isValid(s):
    stack = []
    m = {')':'(', ']':'[', '}':'{'}
    for c in s:
        if c in m:
            if not stack or stack.pop() != m[c]: return False
        else: stack.append(c)
    return not stack
```

140. Minimum Window Substring.

```
Python
def minWindow(s, t):
    if not t or not s: return ""
    countT = Counter(t); window = {}
    have, need = 0, len(countT)
    res, resLen = [-1, -1], float("inf")
    l = 0
    for r in range(len(s)):
        c = s[r]
        window[c] = window.get(c, 0) + 1
        if c in countT and window[c] == countT[c]: have += 1
        while have == need:
            if (r - l + 1) < resLen:
                res = [l, r]; resLen = (r - l + 1)
            window[s[l]] -= 1
            if s[l] in countT and window[s[l]] < countT[s[l]]: have -= 1
            l += 1
    l, r = res
    return s[l:r+1] if resLen != float("inf") else ""
```

141. Encode and Decode Strings.

```
Python
def encode(strs):
    return "".join(str(len(s)) + "#" + s for s in strs)
```

```
def decode(s):
    res, i = [], 0
    while i < len(s):
        j = i
        while s[j] != '#': j += 1
        length = int(s[i:j])
        res.append(s[j+1 : j+1+length])
        i = j + 1 + length
    return res
```

142. Longest Repeating Character Replacement.

```
Python
def characterReplacement(s, k):
    count = {}; res = 0; l = 0; maxf = 0
    for r in range(len(s)):
        count[s[r]] = count.get(s[r], 0) + 1
        maxf = max(maxf, count[s[r]])
        while (r - l + 1) - maxf > k:
            count[s[l]] -= 1; l += 1
        res = max(res, r - l + 1)
    return res
```

143. Valid Palindrome II.

```
Python
def validPalindrome(s):
    l, r = 0, len(s)-1
    while l < r:
        if s[l] != s[r]:
            skipL, skipR = s[l+1:r+1], s[l:r]
            return (skipL == skipL[::-1] or skipR == skipR[::-1])
        l += 1; r -= 1
    return True
```

144. Find All Anagrams.

```
Python
def findAnagrams(s, p):
    pCount, sCount = Counter(p), Counter(s[:len(p)-1])
    res = []
    for i in range(len(p)-1, len(s)):
        sCount[s[i]] += 1
        if sCount == pCount: res.append(i-len(p)+1)
        sCount[s[i-len(p)+1]] -= 1
        if sCount[s[i-len(p)+1]] == 0: del sCount[s[i-len(p)+1]]
    return res
```

145. Count Palindromic Substrings.

Python
```
def countSubstrings(s):
    res = 0
    for i in range(len(s)):
        # Odd & Even
        for l, r in [(i, i), (i, i+1)]:
            while l >= 0 and r < len(s) and s[l] == s[r]:
                res += 1; l -= 1; r += 1
    return res
```

146. Decode Ways.

Python
```
def numDecodings(s):
    dp = {len(s): 1}
    for i in range(len(s) - 1, -1, -1):
        if s[i] == "0": dp[i] = 0
        else: dp[i] = dp[i + 1]
        if (i + 1 < len(s) and (s[i] == "1" or (s[i] == "2" and s[i + 1] in "0123456"))):
            dp[i] += dp[i + 2]
    return dp[0]
```

147. Word Break.

Python
```
def wordBreak(s, wordDict):
    dp = [False] * (len(s) + 1); dp[len(s)] = True
    for i in range(len(s) - 1, -1, -1):
        for w in wordDict:
            if (i + len(w)) <= len(s) and s[i : i + len(w)] == w:
                dp[i] = dp[i + len(w)]
            if dp[i]: break
    return dp[0]
```

148. Wildcard Matching.

```
Python
def isMatch(s, p):
    dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
    dp[0][0] = True
    for j in range(1, len(p) + 1):
        if p[j-1] == '*': dp[0][j] = dp[0][j-1]
    for i in range(1, len(s) + 1):
        for j in range(1, len(p) + 1):
            if p[j-1] == '*': dp[i][j] = dp[i-1][j] or dp[i][j-1]
            elif p[j-1] == '?' or s[i-1] == p[j-1]: dp[i][j] = dp[i-1][j-1]
    return dp[-1][-1]
```

149. Regular Expression Matching.

```
Python
def isMatch(s, p):
    cache = {}
    def dfs(i, j):
        if (i, j) in cache: return cache[(i, j)]
        if i >= len(s) and j >= len(p): return True
        if j >= len(p): return False
        match = i < len(s) and (s[i] == p[j] or p[j] == ".")
        if (j + 1) < len(p) and p[j + 1] == "*":
            cache[(i, j)] = (dfs(i, j + 2) or (match and dfs(i + 1, j)))
            return cache[(i, j)]
        if match:
            cache[(i, j)] = dfs(i + 1, j + 1)
            return cache[(i, j)]
        cache[(i, j)] = False
        return False
    return dfs(0, 0)
```

150. Integer to English.

```
Python
def numberToWords(num):
    if num == 0: return "Zero"
    map = {1000000000: "Billion", 1000000: "Million", 1000: "Thousand", 100: "Hundred", 90:
"Ninety", 80: "Eighty", 70: "Seventy", 60: "Sixty", 50: "Fifty", 40: "Forty", 30: "Thirty", 20:
"Twenty", 19: "Nineteen", 18: "Eighteen", 17: "Seventeen", 16: "Sixteen", 15: "Fifteen", 14:
```

```
"Fourteen", 13: "Thirteen", 12: "Twelve", 11: "Eleven", 10: "Ten", 9: "Nine", 8: "Eight", 7: "Seven",
6: "Six", 5: "Five", 4: "Four", 3: "Three", 2: "Two", 1: "One"}
    def get_words(n):
        for val, word in map.items():
            if n >= val:
                prefix = get_words(n // val) + " " if n >= 100 else ""
                suffix = " " + get_words(n % val) if n % val else ""
                return prefix + word + suffix
        return ""
    return get_words(num)
```

Part 10: Trees & Graphs (Questions 151-175)

151. Max Depth Binary Tree.

Python
```
def maxDepth(root):
    if not root: return 0
    return 1 + max(maxDepth(root.left), maxDepth(root.right))
```

152. Same Tree.

Python
```
def isSameTree(p, q):
    if not p and not q: return True
    if not p or not q or p.val != q.val: return False
    return isSameTree(p.left, q.left) and isSameTree(p.right, q.right)
```

153. Invert Tree.

Python
```
def invertTree(root):
    if root:
        root.left, root.right = invertTree(root.right), invertTree(root.left)
    return root
```

154. Level Order Traversal.

Python
```
def levelOrder(root):
    res, q = [], [root]
    while q and root:
        res.append([n.val for n in q])
        q = [child for n in q for child in (n.left, n.right) if child]
    return res
```

155. Subtree of Another Tree.

```
Python
def isSubtree(s, t):
    if not t: return True
    if not s: return False
    if isSameTree(s, t): return True
    return isSubtree(s.left, t) or isSubtree(s.right, t)
```

156. LCA of BST.

```
Python
def lowestCommonAncestor(root, p, q):
    while root:
        if p.val < root.val and q.val < root.val: root = root.left
        elif p.val > root.val and q.val > root.val: root = root.right
        else: return root
```

157. Validate BST.

```
Python
def isValidBST(root, low=float('-inf'), high=float('inf')):
    if not root: return True
    if not (low < root.val < high): return False
    return isValidBST(root.left, low, root.val) and isValidBST(root.right, root.val, high)
```

158. Kth Smallest in BST.

```
Python
def kthSmallest(root, k):
    stack = []
    while True:
        while root: stack.append(root); root = root.left
        root = stack.pop()
        k -= 1
        if k == 0: return root.val
        root = root.right
```

159. Tree from Preorder/Inorder.

Python
```
def buildTree(preorder, inorder):
    if not preorder or not inorder: return None
    root = TreeNode(preorder[0])
    mid = inorder.index(preorder[0])
    root.left = buildTree(preorder[1:mid+1], inorder[:mid])
    root.right = buildTree(preorder[mid+1:], inorder[mid+1:])
    return root
```

160. Max Path Sum.

Python
```
def maxPathSum(root):
    res = [root.val]
    def dfs(root):
        if not root: return 0
        leftMax = max(dfs(root.left), 0)
        rightMax = max(dfs(root.right), 0)
        res[0] = max(res[0], root.val + leftMax + rightMax)
        return root.val + max(leftMax, rightMax)
    dfs(root)
    return res[0]
```

161. Serialize/Deserialize.

Python
```
def serialize(root):
    res = []
    def dfs(node):
        if not node: res.append("N"); return
        res.append(str(node.val))
        dfs(node.left); dfs(node.right)
    dfs(root); return ",".join(res)
def deserialize(data):
    vals = iter(data.split(","))
    def dfs():
        val = next(vals)
        if val == "N": return None
        node = TreeNode(int(val))
        node.left = dfs(); node.right = dfs()
        return node
    return dfs()
```

162. Number of Islands.

```
Python
def numIslands(grid):
    if not grid: return 0
    count = 0
    def dfs(i, j):
        if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] != '1': return
        grid[i][j] = '#'
        dfs(i+1, j); dfs(i-1, j); dfs(i, j+1); dfs(i, j-1)
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1': dfs(i, j); count += 1
    return count
```

163. Clone Graph.

```
Python
def cloneGraph(node):
    oldToNew = {}
    def dfs(node):
        if node in oldToNew: return oldToNew[node]
        copy = Node(node.val)
        oldToNew[node] = copy
        for nei in node.neighbors: copy.neighbors.append(dfs(nei))
        return copy
    return dfs(node) if node else None
```

164. Pacific Atlantic.

```
Python
def pacificAtlantic(heights):
    ROWS, COLS = len(heights), len(heights[0])
    pac, atl = set(), set()
    def dfs(r, c, visit, prevHeight):
        if ((r, c) in visit or r < 0 or c < 0 or r == ROWS or c == COLS or heights[r][c] < prevHeight):
return
```

```
        visit.add((r, c))
        dfs(r+1, c, visit, heights[r][c]); dfs(r-1, c, visit, heights[r][c])
        dfs(r, c+1, visit, heights[r][c]); dfs(r, c-1, visit, heights[r][c])
    for c in range(COLS): dfs(0, c, pac, heights[0][c]); dfs(ROWS-1, c, atl, heights[ROWS-1][c])
    for r in range(ROWS): dfs(r, 0, pac, heights[r][0]); dfs(r, COLS-1, atl, heights[r][COLS-1])
    return list(pac & atl)
```

165. Course Schedule.

```
Python
def canFinish(numCourses, prerequisites):
    preMap = {i: [] for i in range(numCourses)}
    for crs, pre in prerequisites: preMap[crs].append(pre)
    visit = set()
    def dfs(crs):
        if crs in visit: return False
        if preMap[crs] == []: return True
        visit.add(crs)
        for pre in preMap[crs]:
            if not dfs(pre): return False
        visit.remove(crs)
        preMap[crs] = []
        return True
    for c in range(numCourses):
        if not dfs(c): return False
    return True
```

166. Graph Valid Tree. *Code:* (Same as Number of Connected Components but check `len(edges) == n-1`).

167. Number Connected Components.

```
Python
def countComponents(n, edges):
    par = [i for i in range(n)]
    def find(n):
        res = n
        while res != par[res]: res = par[res]
        return res
    def union(n1, n2):
        p1, p2 = find(n1), find(n2)
        if p1 == p2: return 0
        par[p1] = p2; return 1
    return n - sum(union(n1, n2) for n1, n2 in edges)
```

168. Word Ladder.

Python
```
def ladderLength(beginWord, endWord, wordList):
    if endWord not in wordList: return 0
    nei = defaultdict(list)
    wordList.append(beginWord)
    for word in wordList:
        for j in range(len(word)):
            pattern = word[:j] + "*" + word[j + 1:]
            nei[pattern].append(word)
    visit, q = set([beginWord]), deque([beginWord])
    res = 1
    while q:
        for i in range(len(q)):
            word = q.popleft()
            if word == endWord: return res
            for j in range(len(word)):
                pattern = word[:j] + "*" + word[j + 1:]
                for neiWord in nei[pattern]:
                    if neiWord not in visit:
                        visit.add(neiWord); q.append(neiWord)
        res += 1
    return 0
```

169. Alien Dictionary.

Python
```
def alienOrder(words):
    adj = {c:set() for w in words for c in w}
    for i in range(len(words)-1):
        w1, w2 = words[i], words[i+1]
        minLen = min(len(w1), len(w2))
        if len(w1) > len(w2) and w1[:minLen] == w2[:minLen]: return ""
        for j in range(minLen):
            if w1[j] != w2[j]: adj[w1[j]].add(w2[j]); break
    visit, res = {}, []
    def dfs(c):
        if c in visit: return visit[c]
        visit[c] = True
        for nei in adj[c]:
            if dfs(nei): return True
        visit[c] = False
        res.append(c)
    for c in adj:
        if dfs(c): return ""
    return "".join(res[::-1])
```

170. Implement Trie.

Python
```
class Trie:
    def __init__(self): self.root = {}
    def insert(self, word):
        cur = self.root
        for c in word:
            if c not in cur: cur[c] = {}
            cur = cur[c]
        cur['*'] = True
    def search(self, word):
        cur = self.root
        for c in word:
            if c not in cur: return False
            cur = cur[c]
        return '*' in cur
```

171. Add and Search Word (Wildcard).

Python
```
class WordDictionary:
    def __init__(self): self.root = {}
    def addWord(self, word):
        cur = self.root
        for c in word:
            if c not in cur: cur[c] = {}
            cur = cur[c]
        cur['*'] = True
    def search(self, word):
        def dfs(j, root):
            cur = root
            for i in range(j, len(word)):
                c = word[i]
                if c == ".":
                    for child in cur.values():
```

```
                if child != True and dfs(i+1, child): return True
            return False
        if c not in cur: return False
        cur = cur[c]
    return '*' in cur
return dfs(0, self.root)
```

172. Word Search II. *Code:* (Combine Trie insert and grid DFS).

173. Diameter of Tree.

Python
```
def diameterOfBinaryTree(root):
    res = [0]
    def dfs(root):
        if not root: return -1
        left = dfs(root.left); right = dfs(root.right)
        res[0] = max(res[0], 2 + left + right)
        return 1 + max(left, right)
    dfs(root); return res[0]
```

174. Right Side View.

Python
```
def rightSideView(root):
    res, q = [], deque([root])
    while q:
        rightSide = None
        for i in range(len(q)):
            node = q.popleft()
            if node: rightSide = node; q.append(node.left); q.append(node.right)
        if rightSide: res.append(rightSide.val)
    return res
```

175. Flatten Tree to List.

Python
```
def flatten(root):
    def dfs(root):
        if not root: return None
        leftTail = dfs(root.left); rightTail = dfs(root.right)
        if root.left:
```

```
        leftTail.right = root.right
        root.right = root.left
        root.left = None
    return rightTail or leftTail or root
dfs(root)
```

Part 11: DP & Backtracking (Questions 176-200)

176. Climbing Stairs.

```
Python
def climbStairs(n):
    one, two = 1, 1
    for i in range(n-1):
        temp = one; one = one + two; two = temp
    return one
```

177. Coin Change.

```
Python
def coinChange(coins, amount):
    dp = [amount + 1] * (amount + 1)
    dp[0] = 0
    for a in range(1, amount + 1):
        for c in coins:
            if a - c >= 0: dp[a] = min(dp[a], 1 + dp[a - c])
    return dp[amount] if dp[amount] != amount + 1 else -1
```

178. Longest Increasing Subsequence.

```
Python
def lengthOfLIS(nums):
    LIS = [1] * len(nums)
    for i in range(len(nums) - 1, -1, -1):
        for j in range(i + 1, len(nums)):
            if nums[i] < nums[j]: LIS[i] = max(LIS[i], 1 + LIS[j])
    return max(LIS)
```

179. LCS.

```python
def longestCommonSubsequence(text1, text2):
    dp = [[0 for j in range(len(text2) + 1)] for i in range(len(text1) + 1)]
    for i in range(len(text1) - 1, -1, -1):
        for j in range(len(text2) - 1, -1, -1):
            if text1[i] == text2[j]: dp[i][j] = 1 + dp[i + 1][j + 1]
            else: dp[i][j] = max(dp[i][j + 1], dp[i + 1][j])
    return dp[0][0]
```

180. House Robber.

```python
def rob(nums):
    rob1, rob2 = 0, 0
    for n in nums:
        temp = max(n + rob1, rob2)
        rob1 = rob2; rob2 = temp
    return rob2
```

181. House Robber II.

```python
def rob(nums):
    return max(nums[0], helper(nums[1:]), helper(nums[:-1]))
def helper(nums):
    rob1, rob2 = 0, 0
    for n in nums:
        newRob = max(rob1 + n, rob2); rob1 = rob2; rob2 = newRob
    return rob2
```

182. Unique Paths.

```python
def uniquePaths(m, n):
    row = [1] * n
    for i in range(m - 1):
        newRow = [1] * n
        for j in range(n - 2, -1, -1):
            newRow[j] = newRow[j + 1] + row[j]
        row = newRow
    return row[0]
```

183. Jump Game.

```
Python
def canJump(nums):
    goal = len(nums) - 1
    for i in range(len(nums) - 1, -1, -1):
        if i + nums[i] >= goal: goal = i
    return goal == 0
```

184. Combination Sum.

```
Python
def combinationSum(candidates, target):
    res = []
    def dfs(i, cur, total):
        if total == target: res.append(cur.copy()); return
        if i >= len(candidates) or total > target: return
        cur.append(candidates[i])
        dfs(i, cur, total + candidates[i])
        cur.pop()
        dfs(i + 1, cur, total)
    dfs(0, [], 0); return res
```

185. Permutations.

```
Python
def permute(nums):
    if len(nums) == 0: return [[]]
    perms = permute(nums[1:])
    res = []
    for p in perms:
        for i in range(len(p) + 1):
            p_copy = p.copy()
            p_copy.insert(i, nums[0])
            res.append(p_copy)
    return res
```

186. Subsets.

```
Python
def subsets(nums):
    res = []
    subset = []
    def dfs(i):
        if i >= len(nums): res.append(subset.copy()); return
        subset.append(nums[i])
        dfs(i + 1)
        subset.pop()
        dfs(i + 1)
    dfs(0); return res
```

187. Word Search.

Python
```
def exist(board, word):
    ROWS, COLS = len(board), len(board[0])
    path = set()
    def dfs(r, c, i):
        if i == len(word): return True
        if (r < 0 or c < 0 or r >= ROWS or c >= COLS or word[i] != board[r][c] or (r, c) in path): return
False
        path.add((r, c))
        res = (dfs(r+1,c,i+1) or dfs(r-1,c,i+1) or dfs(r,c+1,i+1) or dfs(r,c-1,i+1))
        path.remove((r, c))
        return res
    for r in range(ROWS):
        for c in range(COLS):
            if dfs(r, c, 0): return True
    return False
```

188. Palindrome Partitioning.

Python
```
def partition(s):
    res, part = [], []
    def dfs(i):
        if i >= len(s): res.append(part.copy()); return
        for j in range(i, len(s)):
            if isPali(s, i, j):
                part.append(s[i:j+1]); dfs(j+1); part.pop()
    dfs(0); return res
def isPali(s, l, r):
    while l < r:
        if s[l] != s[r]: return False
        l, r = l+1, r-1
    return True
```

189. N-Queens.

Python
```
def solveNQueens(n):
    col, posDiag, negDiag = set(), set(), set()
    res = [], board = [["."] * n for _ in range(n)]
    def backtrack(r):
        if r == n: res.append(["".join(row) for row in board]); return
        for c in range(n):
            if c in col or (r+c) in posDiag or (r-c) in negDiag: continue
            col.add(c); posDiag.add(r+c); negDiag.add(r-c); board[r][c] = "Q"
            backtrack(r + 1)
            col.remove(c); posDiag.remove(r+c); negDiag.remove(r-c); board[r][c] = "."
    backtrack(0); return res
```

190. Edit Distance.

Python
```
def minDistance(word1, word2):
    cache = [[float("inf")] * (len(word2) + 1) for i in range(len(word1) + 1)]
    for j in range(len(word2) + 1): cache[len(word1)][j] = len(word2) - j
    for i in range(len(word1) + 1): cache[i][len(word2)] = len(word1) - i
    for i in range(len(word1) - 1, -1, -1):
        for j in range(len(word2) - 1, -1, -1):
            if word1[i] == word2[j]: cache[i][j] = cache[i + 1][j + 1]
            else: cache[i][j] = 1 + min(cache[i + 1][j], cache[i][j + 1], cache[i + 1][j + 1])
    return cache[0][0]
```

191. Maximal Square.

Python
```
def maximalSquare(matrix):
    ROWS, COLS = len(matrix), len(matrix[0])
    cache = {}
    def helper(r, c):
        if r >= ROWS or c >= COLS: return 0
        if (r, c) not in cache:
            down = helper(r + 1, c); right = helper(r, c + 1); diag = helper(r + 1, c + 1)
            cache[(r, c)] = 0
            if matrix[r][c] == "1": cache[(r, c)] = 1 + min(down, right, diag)
        return cache[(r, c)]
    helper(0, 0)
    return max(cache.values()) ** 2 if cache else 0
```

192. Partition Equal Subset Sum.

Python
```
def canPartition(nums):
    if sum(nums) % 2: return False
    dp = set(); dp.add(0)
    target = sum(nums) // 2
    for i in range(len(nums) - 1, -1, -1):
        nextDP = set()
        for t in dp:
            nextDP.add(t + nums[i]); nextDP.add(t)
        dp = nextDP
    return True if target in dp else False
```

193. Target Sum.

Python
```
def findTargetSumWays(nums, target):
    dp = {0: 1}
    for n in nums:
        nextDP = defaultdict(int)
        for curSum, count in dp.items():
            nextDP[curSum + n] += count
            nextDP[curSum - n] += count
        dp = nextDP
    return dp[target]
```

194. Burst Balloons.

Python
```
def maxCoins(nums):
    nums = [1] + nums + [1]
    dp = {}
    def dfs(l, r):
        if l > r: return 0
        if (l, r) in dp: return dp[(l, r)]
        dp[(l, r)] = 0
        for i in range(l, r + 1):
            coins = nums[l-1] * nums[i] * nums[r+1]
            coins += dfs(l, i-1) + dfs(i+1, r)
            dp[(l, r)] = max(dp[(l, r)], coins)
        return dp[(l, r)]
    return dfs(1, len(nums) - 2)
```

195. Regular Expression (Repeated - see Q149). *Code already provided in Q149.*

196. Interleaving String.

```
Python
def isInterleave(s1, s2, s3):
    if len(s1) + len(s2) != len(s3): return False
    dp = [[False] * (len(s2) + 1) for i in range(len(s1) + 1)]
    dp[len(s1)][len(s2)] = True
    for i in range(len(s1), -1, -1):
        for j in range(len(s2), -1, -1):
            if i < len(s1) and s1[i] == s3[i + j] and dp[i + 1][j]: dp[i][j] = True
            if j < len(s2) and s2[j] == s3[i + j] and dp[i][j + 1]: dp[i][j] = True
    return dp[0][0]
```

197. Distinct Subsequences.

```
Python
def numDistinct(s, t):
    cache = {}
    def dfs(i, j):
        if j == len(t): return 1
        if i == len(s): return 0
        if (i, j) in cache: return cache[(i, j)]
        if s[i] == t[j]: cache[(i, j)] = dfs(i + 1, j + 1) + dfs(i + 1, j)
        else: cache[(i, j)] = dfs(i + 1, j)
        return cache[(i, j)]
    return dfs(0, 0)
```

198. Letter Combinations Phone Number.

```
Python
def letterCombinations(digits):
    res = []
    digitToChar = {"2":"abc", "3":"def", "4":"ghi", "5":"jkl", "6":"mno", "7":"qprs", "8":"tuv",
"9":"wxyz"}
```

```
    def backtrack(i, curStr):
        if len(curStr) == len(digits): res.append(curStr); return
        for c in digitToChar[digits[i]]:
            backtrack(i + 1, curStr + c)
    if digits: backtrack(0, "")
    return res
```

199. Generate Parentheses.

```
Python
def generateParenthesis(n):
    stack, res = [], []
    def backtrack(openN, closedN):
        if openN == closedN == n: res.append("".join(stack)); return
        if openN < n: stack.append("("); backtrack(openN + 1, closedN); stack.pop()
        if closedN < openN: stack.append(")"); backtrack(openN, closedN + 1); stack.pop()
    backtrack(0, 0); return res
```

200. Sudoku Solver.

```
Python
def solveSudoku(board):
    def isValid(r, c, k):
        for i in range(9):
            if board[i][c] == k or board[r][i] == k: return False
            if board[3*(r//3) + i//3][3*(c//3) + i%3] == k: return False
        return True
    def solve():
        for r in range(9):
            for c in range(9):
                if board[r][c] == '.':
                    for k in map(str, range(1, 10)):
                        if isValid(r, c, k):
                            board[r][c] = k
                            if solve(): return True
                            board[r][c] = '.'
                    return False
        return True
    solve()
```