100+ Python Programming Problems with Solutions: From Basics to Advanced

Easy Python Problems (with Solutions)

1. Print "Hello, World!"

```python
print("Hello, World!")
```

2. Swap two variables

```python
a, b = 5, 10
a, b = b, a
print(a, b)
```

3. Check even or odd

```python
n = 7
if n % 2 == 0:
    print("Even")
else:
    print("Odd")
```

4. Find factorial (loop)

```python
n = 5
fact = 1
for i in range(1, n+1):
    fact *= i
print(fact)
```

5. Sum of first N natural numbers

```python
n = 10
total = n * (n + 1) // 2
print(total)
```

6. Reverse a string

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

7. Palindrome check

```
s = "madam"
if s == s[::-1]:
    print("Palindrome")
else:
    print("Not Palindrome")
```

8. Fibonacci series (first 10 numbers)

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

9. Find max of three numbers

```
a, b, c = 3, 7, 5
print(max(a, b, c))
```

10. Check prime number

```
n = 29
is_prime = True
for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        is_prime = False
        break
print("Prime" if is_prime else "Not Prime")
```

11. Find sum of digits

```
n = 1234
print(sum(int(d) for d in str(n)))
```

12. Find largest element in a list

```
nums = [4, 9, 2, 15, 7]
print(max(nums))
```

13. Count vowels in a string

```
s = "programming"
vowels = "aeiou"
count = sum(1 for ch in s if ch in vowels)
print(count)
```

14. Multiplication table of a number

```python
n = 5
for i in range(1, 11):
    print(n, "x", i, "=", n*i)
```

15. Check leap year

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

16. Convert Celsius to Fahrenheit

```python
c = 37
f = (c * 9/5) + 32
print(f)
```

17. Find ASCII value of a character

```python
ch = 'A'
print(ord(ch))
```

18. Reverse a number

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

19. Simple calculator (add, sub, mul, div)

```python
a, b = 10, 5
print("Add:", a+b)
print("Sub:", a-b)
print("Mul:", a*b)
print("Div:", a/b)
```

20. Count occurrences of a character in string

```python
s = "programming"
print(s.count("m"))
```

21. Find smallest element in a list

```python
nums = [4, 9, 2, 15, 7]
print(min(nums))
```

22. Check Armstrong number (3-digit)

```python
n = 153
total = sum(int(d)**3 for d in str(n))
print("Armstrong" if total == n else "Not Armstrong")
```

23. Print all divisors of a number

```python
n = 12
for i in range(1, n+1):
    if n % i == 0:
        print(i, end=" ")
```

24. Find length of string without len()

```python
s = "python"
count = 0
for _ in s:
    count += 1
print(count)
```

25. Print ASCII characters from A to Z

```python
for i in range(65, 91):
    print(chr(i), end=" ")
```

26. Sum of elements in a list

```python
nums = [10, 20, 30, 40]
print(sum(nums))
```

27. Remove duplicates from list

```python
nums = [1, 2, 2, 3, 4, 4, 5]
print(list(set(nums)))
```

28. Convert decimal to binary

```python
n = 13
print(bin(n)[2:])
```

29. Convert binary to decimal

```python
b = "1101"
print(int(b, 2))
```

30. Generate random number (1–100)

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

Medium Python Problems (with Solutions)

31. Find GCD (Euclidean Algorithm)

```python
 import math
a, b = 48, 18
print(math.gcd(a, b))
```

32. Find LCM of two numbers

```python
import math
a, b = 12, 15
print(abs(a*b) // math.gcd(a, b))
```

33. Binary to Decimal

```python
b = "1011"
print(int(b, 2))
```

34. Decimal to Binary

```python
n = 13
print(bin(n)[2:])
```

35. Armstrong number (any digits)

```python
 n = 9474
power = len(str(n))
print(sum(int(d)**power for d in str(n)) == n)
```

36. Count words in a string

```python
 s = "I love Python programming"
print(len(s.split()))
```

37. Check if two strings are anagrams

```python
a, b = "listen", "silent"
print(sorted(a) == sorted(b))
```

38. Find second largest element in list

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

39. Matrix addition

```
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
result = [[A[i][j] + B[i][j] for j in range(2)] for i in range(2)]
print(result)
```

40. Transpose of a matrix

```
A = [[1, 2, 3], [4, 5, 6]]
print(list(zip(*A)))
```

41. Factorial using recursion

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

42. Fibonacci using recursion

```
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

for i in range(10):
    print(fib(i), end=" ")
```

43. Find prime factors of a number

```
n = 84
factors = []
i = 2
while n > 1:
    if n % i == 0:
        factors.append(i)
        n //= i
    else:
        i += 1
print(factors)
```

44. Bubble sort

```python
arr = [64, 25, 12, 22, 11]
for i in range(len(arr)):
    for j in range(len(arr)-i-1):
        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
print(arr)
```

45. Insertion sort

```python
arr = [12, 11, 13, 5, 6]
for i in range(1, len(arr)):
    key = arr[i]
    j = i-1
    while j >= 0 and key < arr[j]:
        arr[j+1] = arr[j]
        j -= 1
    arr[j+1] = key
print(arr)
```

46. Linear search

```python
arr = [10, 20, 30, 40, 50]
x = 30
for i in range(len(arr)):
    if arr[i] == x:
        print("Found at index", i)
        break
```

47. Binary search

```python
arr = [10, 20, 30, 40, 50]
x = 40
low, high = 0, len(arr)-1
while low <= high:
    mid = (low+high)//2
    if arr[mid] == x:
        print("Found at index", mid)
        break
    elif arr[mid] < x:
        low = mid+1
    else:
        high = mid-1
```

48. Remove duplicates from string

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

49. Frequency of characters in a string

```
s = "hello world"
freq = {}
for ch in s:
    freq[ch] = freq.get(ch, 0) + 1
print(freq)
```

50. Reverse words in a string

```
s = "I love Python"
print(" ".join(s.split()[::-1]))
```

51. Check pangram (contains all letters)

```
import string
s = "The quick brown fox jumps over the lazy dog"
print(set(string.ascii_lowercase) <= set(s.lower()))
```

52. Find common elements in two lists

```
a = [1, 2, 3, 4]
b = [3, 4, 5, 6]
print(list(set(a) & set(b)))
```

53. Find unique elements in list

```
nums = [1, 2, 2, 3, 4, 4, 5]
unique = [x for x in nums if nums.count(x) == 1]
print(unique)
```

54. Merge two sorted lists

```
a = [1, 3, 5]
b = [2, 4, 6]
print(sorted(a+b))
```

55. Dictionary sorting by values

```
d = {'a': 3, 'b': 1, 'c': 2}
print(dict(sorted(d.items(), key=lambda x: x[1])))
```

56. Count vowels and consonants

```
s = "Python Programming"
vowels = "aeiouAEIOU"
v = sum(1 for ch in s if ch in vowels)
c = sum(1 for ch in s if ch.isalpha() and ch not in vowels)
print("Vowels:", v, "Consonants:", c)
```

57. Find longest word in a string

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

58. Check if string is numeric

```
s = "12345"
print(s.isdigit())
```

59. File read and write

```
with open("sample.txt", "w") as f:
    f.write("Hello Python\n")

with open("sample.txt", "r") as f:
    print(f.read())
```

60. Exception handling

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Division by zero is not allowed")
```

61. Simple calculator using functions

```
def add(a,b): return a+b
def sub(a,b): return a-b
def mul(a,b): return a*b
def div(a,b): return a/b if b!=0 else "Error"

print(add(5,3), sub(5,3), mul(5,3), div(5,3))
```

62. List comprehension (squares)

```
squares = [x**2 for x in range(1, 11)]
print(squares)
```

63. Flatten a nested list

```
nested = [[1,2],[3,4],[5,6]]
flat = [x for sub in nested for x in sub]
print(flat)
```

64. Prime numbers in a range

```
primes = []
for n in range(10, 51):
    if all(n % i for i in range(2, int(n**0.5)+1)):
        primes.append(n)
print(primes)
```

65. Find sum of factorials of digits

```
import math
n = 145
print(sum(math.factorial(int(d)) for d in str(n)))
```

66. Check strong number (sum of factorials = number)

```
import math
n = 145
if sum(math.factorial(int(d)) for d in str(n)) == n:
    print("Strong Number")
else:
    print("Not Strong Number")
```

67. Count uppercase, lowercase, digits, special chars

```
s = "Python3.9!"
u = sum(1 for ch in s if ch.isupper())
l = sum(1 for ch in s if ch.islower())
d = sum(1 for ch in s if ch.isdigit())
sc = len(s) - (u+l+d)
print(u, l, d, sc)
```

68. Find HCF of multiple numbers

```
import math
nums = [24, 36, 60]
hcf = math.gcd(nums[0], nums[1])
for i in nums[2:]:
    hcf = math.gcd(hcf, i)
print(hcf)
```

69. Find LCM of multiple numbers

```
import math
nums = [4, 6, 8]
lcm = nums[0]
for i in nums[1:]:
    lcm = abs(lcm*i) // math.gcd(lcm, i)
print(lcm)
```

70. Convert Roman numeral to integer

```
def roman_to_int(s):
    values = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
    total = 0
    prev = 0
    for ch in s[::-1]:
        if values[ch] < prev:
            total -= values[ch]
        else:
            total += values[ch]
        prev = values[ch]
    return total

print(roman_to_int("MCMXCIV"))  # 1994
```

Difficult Python Problems (with Solutions)

71. Check Sudoku validity

```python
def valid_sudoku(board):
    seen = set()
    for i in range(9):
        for j in range(9):
            num = board[i][j]
            if num != '.':
                if (i, num) in seen or (num, j) in seen or (i//3, j//3, num) in seen:
                    return False
                seen |= {(i, num), (num, j), (i//3, j//3, num)}
    return True
```

72. N-Queens Problem (Backtracking)

```python
def solve_nqueens(n):
    res, board = [], [-1]*n

    def valid(r,c):
        for i in range(r):
            if board[i]==c or abs(board[i]-c)==r-i:
                return False
        return True
```

```python
    def dfs(r):
        if r==n:
            res.append(board[:])
            return
        for c in range(n):
            if valid(r,c):
                board[r]=c
                dfs(r+1)

    dfs(0)
    return res

print(len(solve_nqueens(8)))  # 92 solutions
```

73. Knapsack Problem (Dynamic Programming)

```
def knapsack(W, wt, val, n):
    dp = [[0]*(W+1) for _ in range(n+1)]
    for i in range(1, n+1):
        for w in range(1, W+1):
            if wt[i-1] <= w:
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]], dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][W]

print(knapsack(50,[10,20,30],[60,100,120],3))
```

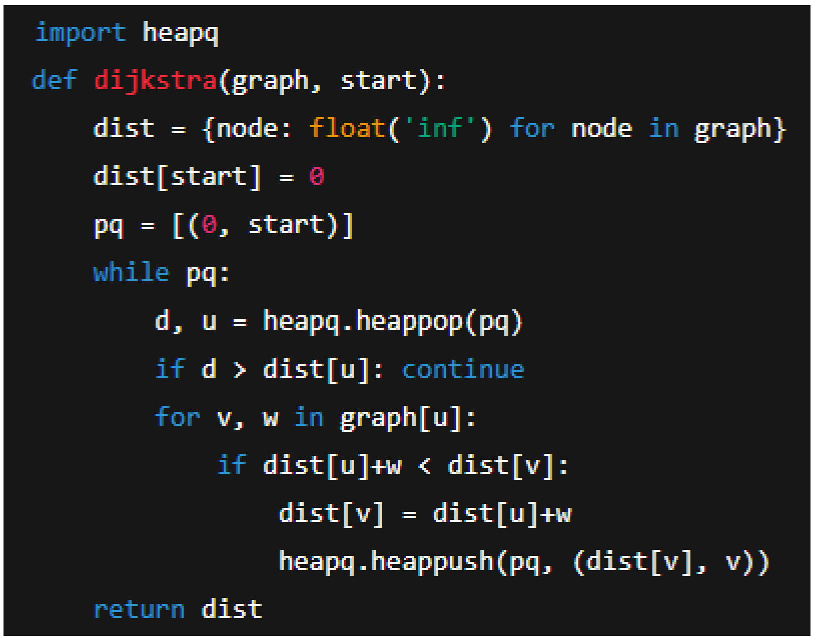
74. Longest Common Subsequence

```
def lcs(X, Y):
    m, n = len(X), len(Y)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if X[i] == Y[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j])
    return dp[m][n]

print(lcs("ABCBDAB","BDCAB"))  # 4
```

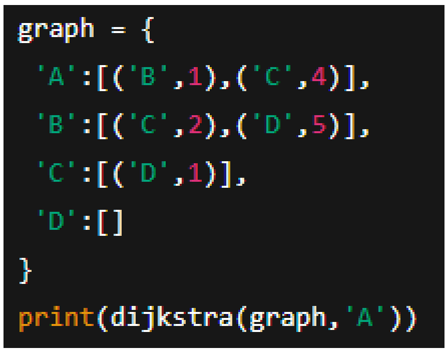
75. Word Break Problem

```
def word_break(s, word_dict):
    dp = [False]*(len(s)+1)
    dp[0] = True
    for i in range(1, len(s)+1):
        for w in word_dict:
            if dp[i-len(w)] and s[i-len(w):i] == w:
                dp[i] = True
    return dp[-1]

print(word_break("leetcode", ["leet","code"]))
```

76. Dijkstra's Algorithm

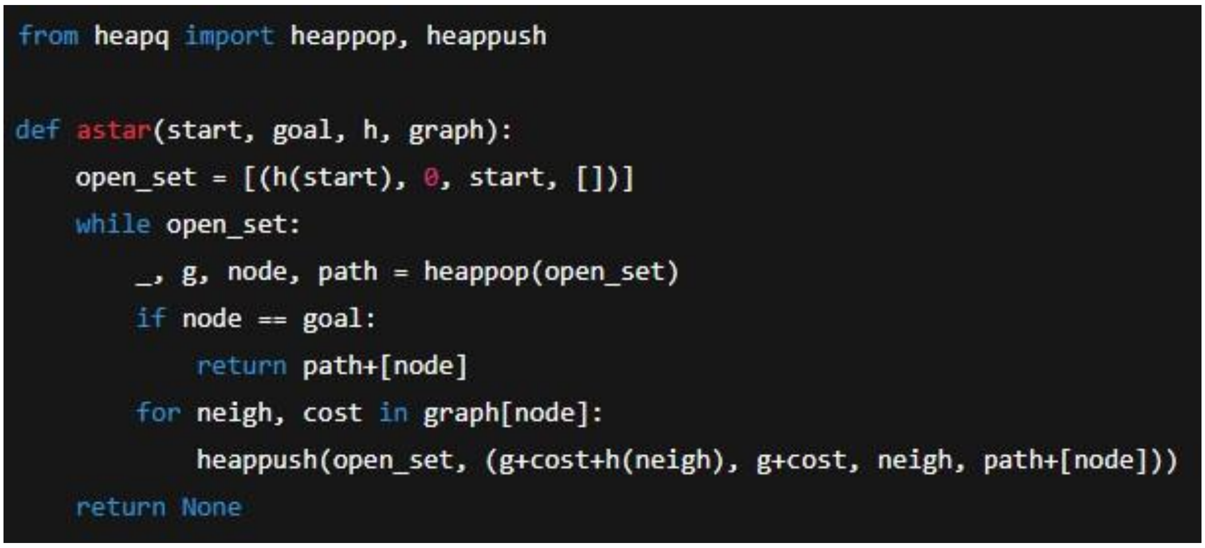


```python
 import heapq
def dijkstra(graph, start):
    dist = {node: float('inf') for node in graph}
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]: continue
        for v, w in graph[u]:
            if dist[u]+w < dist[v]:
                dist[v] = dist[u]+w
                heapq.heappush(pq, (dist[v], v))
    return dist
```



```python
graph = {
 'A':[('B',1),('C',4)],
 'B':[('C',2),('D',5)],
 'C':[('D',1)],
 'D':[]
}
print(dijkstra(graph,'A'))
```

77. A* Search Algorithm



```python
from heapq import heappop, heappush

def astar(start, goal, h, graph):
    open_set = [(h(start), 0, start, [])]
    while open_set:
        _, g, node, path = heappop(open_set)
        if node == goal:
            return path+[node]
        for neigh, cost in graph[node]:
            heappush(open_set, (g+cost+h(neigh), g+cost, neigh, path+[node]))
    return None
```

```python
graph = {
 'A':[('B',1),('C',3)],
 'B':[('D',3)],
 'C':[('D',1)],
 'D':[]
}
h = lambda x: {'A':3,'B':2,'C':1,'D':0}[x]
print(astar('A','D',h,graph))
```

78. Merge Sort

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L, R = arr[:mid], arr[mid:]
        merge_sort(L); merge_sort(R)
        i=j=k=0
        while i<len(L) and j<len(R):
            if L[i]<R[j]: arr[k]=L[i]; i+=1
            else: arr[k]=R[j]; j+=1
            k+=1
        while i<len(L): arr[k]=L[i]; i+=1; k+=1
        while j<len(R): arr[k]=R[j]; j+=1; k+=1

arr = [38,27,43,3,9,82,10]
merge_sort(arr)
print(arr)
```

79. Quick Sort

```python
def quick_sort(arr):
    if len(arr)<=1:
        return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x<pivot]
    mid = [x for x in arr if x==pivot]
    right= [x for x in arr if x>pivot]
    return quick_sort(left)+mid+quick_sort(right)

print(quick_sort([10,7,8,9,1,5]))
```

80. Binary Tree Traversals

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

def inorder(root):
    if root: inorder(root.left); print(root.val,end=" "); inorder(root.right)

root=Node(1); root.left=Node(2); root.right=Node(3)
root.left.left=Node(4); root.left.right=Node(5)
inorder(root)
```

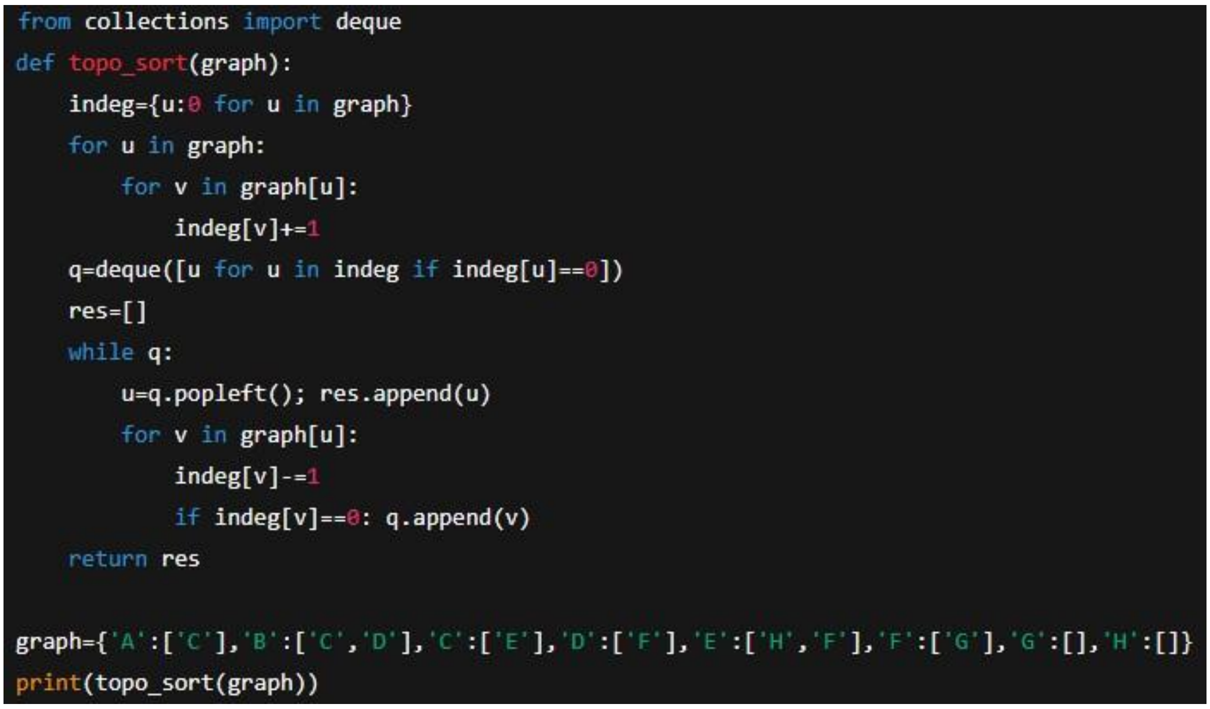
81. Check if Binary Tree is Balanced

```python
def is_balanced(root):
    def height(node):
        if not node: return 0
        l = height(node.left)
        r = height(node.right)
        if l==-1 or r==-1 or abs(l-r)>1: return -1
        return max(l,r)+1
    return height(root)!=-1
```

82. Graph BFS

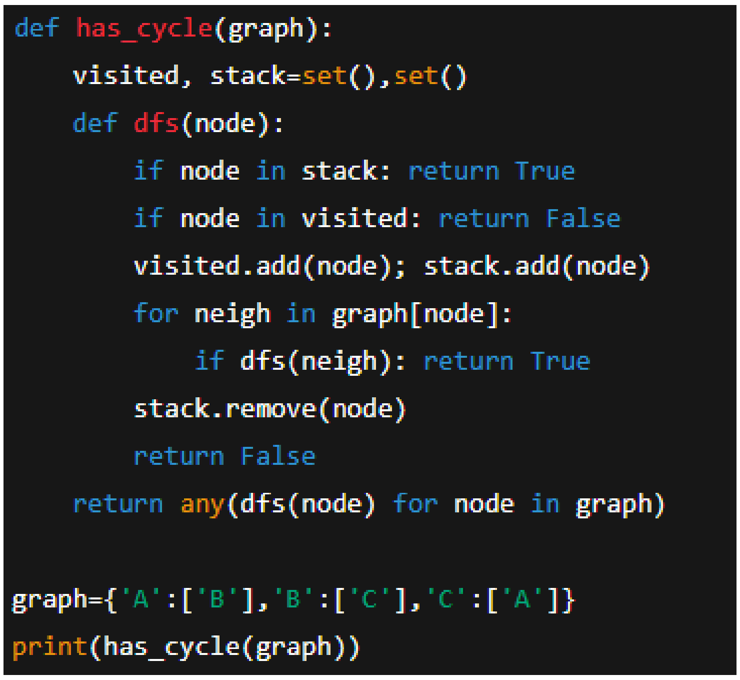
```python
from collections import deque
def bfs(graph, start):
    visited=set([start])
    q=deque([start])
    while q:
        node=q.popleft()
        print(node,end=" ")
        for neigh in graph[node]:
            if neigh not in visited:
                visited.add(neigh); q.append(neigh)

graph={'A':['B','C'],'B':['D'],'C':['E'],'D':[],'E':[]}
bfs(graph,'A')
```

83. Topological Sort (Kahn's Algorithm)



```python
from collections import deque
def topo_sort(graph):
    indeg={u:0 for u in graph}
    for u in graph:
        for v in graph[u]:
            indeg[v]+=1
    q=deque([u for u in indeg if indeg[u]==0])
    res=[]
    while q:
        u=q.popleft(); res.append(u)
        for v in graph[u]:
            indeg[v]-=1
            if indeg[v]==0: q.append(v)
    return res

graph={'A':['C'],'B':['C','D'],'C':['E'],'D':['F'],'E':['H','F'],'F':['G'],'G':[],'H':[]}
print(topo_sort(graph))
```

84. Detect Cycle in Graph (DFS)



```python
def has_cycle(graph):
    visited, stack=set(),set()
    def dfs(node):
        if node in stack: return True
        if node in visited: return False
        visited.add(node); stack.add(node)
        for neigh in graph[node]:
            if dfs(neigh): return True
        stack.remove(node)
        return False
    return any(dfs(node) for node in graph)

graph={'A':['B'],'B':['C'],'C':['A']}
print(has_cycle(graph))
```

85. Trie Implementation

```
class TrieNode:
    def __init__(self):
        self.children={}; self.end=False

class Trie:
    def __init__(self): self.root=TrieNode()
    def insert(self,word):
        node=self.root
        for ch in word:
            if ch not in node.children: node.children[ch]=TrieNode()
            node=node.children[ch]
        node.end=True
```

```
    def search(self,word):
        node=self.root
        for ch in word:
            if ch not in node.children: return False
            node=node.children[ch]
        return node.end

t=Trie()
t.insert("hello")
print(t.search("hello"))
```

86. LRU Cache

```
from collections import OrderedDict
class LRUCache:
    def __init__(self,cap): self.cap=cap; 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,val):
        if key in self.cache: self.cache.move_to_end(key)
        self.cache[key]=val
        if len(self.cache)>self.cap: self.cache.popitem(last=False)

lru=LRUCache(2)
lru.put(1,1); lru.put(2,2)
print(lru.get(1))
lru.put(3,3)
print(lru.get(2))
```

87. Regex-based Email Validator

```
import re
pattern=r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-z]{2,}$'
print(bool(re.match(pattern,"test@example.com")))
```

88. Web Scraping with BeautifulSoup

```
import requests
from bs4 import BeautifulSoup
url="https://example.com"
html=requests.get(url).text
soup=BeautifulSoup(html,"html.parser")
print(soup.title.string)
```

89. Simple REST API with Flask

```
from flask import Flask, jsonify
app=Flask(__name__)

@app.route('/api/hello')
def hello(): return jsonify({"message":"Hello World"})

if __name__=="__main__":
    app.run(debug=True)
```

90. SQLite CRUD Example

```
import sqlite3
con=sqlite3.connect("test.db")
cur=con.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS users(id INTEGER,name TEXT)")
cur.execute("INSERT INTO users VALUES(1,'Alice')")
con.commit()
for row in cur.execute("SELECT * FROM users"):
    print(row)
con.close()
```

91. Multithreading Example

```python
import threading
def task(name):
    for i in range(5):
        print(name,i)

t1=threading.Thread(target=task,args=("A",))
t2=threading.Thread(target=task,args=("B",))
t1.start(); t2.start()
t1.join(); t2.join()
```

92. Producer-Consumer Problem

```python
import threading, queue
q=queue.Queue()
def producer():
    for i in range(5):
        q.put(i); print("Produced",i)
def consumer():
    for i in range(5):
        item=q.get(); print("Consumed",item); q.task_done()
t1=threading.Thread(target=producer)
t2=threading.Thread(target=consumer)
t1.start(); t2.start(); t1.join(); t2.join()
```

93. Simple Chatbot using NLTK

```python
from nltk.chat.util import Chat, reflections
pairs=[(r"hi|hello","Hello!"),(r"how are you","I'm fine, thanks.")]
chat=Chat(pairs,reflections)
chat.converse()
```

94. Image to Grayscale with PIL

```python
from PIL import Image
img=Image.open("sample.jpg").convert("L")
img.save("gray.jpg")
```

95. CSV Data Analysis with Pandas

```
import pandas as pd
df=pd.read_csv("data.csv")
print(df.head())
print(df['column_name'].mean())
```

96. Matplotlib plotting

```
import matplotlib.pyplot as plt
x=[1,2,3,4]; y=[1,4,9,16]
plt.plot(x,y)
plt.show()
```

97. Linear Regression (sklearn)

```
from sklearn.linear_model import LinearRegression
import numpy as np
X=np.array([[1],[2],[3],[4]])
y=np.array([2,4,6,8])
model=LinearRegression().fit(X,y)
print(model.predict([[5]]))
```

98. K-means Clustering

```
from sklearn.cluster import KMeans
import numpy as np
X=np.array([[1,2],[1,4],[1,0],[10,2],[10,4],[10,0]])
kmeans=KMeans(n_clusters=2,random_state=0).fit(X)
print(kmeans.labels_)
```

99. Simple Neural Network (NumPy)

```
import numpy as np
X=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([[0],[1],[1],[0]])
W1=np.random.rand(2,2); W2=np.random.rand(2,1)
for _ in range(10000):
    l1=1/(1+np.exp(-(X@W1)))
    l2=1/(1+np.exp(-(l1@W2)))
print(np.round(l2,2))
```

100. Web Automation with Selenium

```
from selenium import webdriver
driver=webdriver.Chrome()
driver.get("https://example.com")
print(driver.title)
driver.quit()
```

Extra 20 Python Problems (Advanced & Special Topics)

101. Generator function (yield)

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

for num in countdown(5):
    print(num, end=" ")
```

102. Custom iterator

```
class Counter:
    def __init__(self, low, high):
        self.current = low
        self.high = high
    def __iter__(self):
        return self
    def __next__(self):
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1

for i in Counter(1, 5):
    print(i)
```

103. Decorator for logging function calls

```
def logger(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}")
        return func(*args, **kwargs)
    return wrapper

@logger
def greet(name):
    return f"Hello {name}"

print(greet("Krish"))
```

104. Regular expressions: validate email

```python
import re
email = "test@example.com"
pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
print(bool(re.match(pattern, email)))
```

105. Regular expressions: extract numbers

```python
import re
s = "The price is 120 dollars and 45 cents"
print(re.findall(r'\d+', s))
```

106. Context manager (custom class)

```python
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode
    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file.close()

with FileManager("test.txt", "w") as f:
    f.write("Hello Context Manager")
```

107. Using with and contextlib

```python
from contextlib import contextmanager

@contextmanager
def open_file(name, mode):
    f = open(name, mode)
    yield f
    f.close()

with open_file("demo.txt", "w") as f:
    f.write("Using contextlib!")
```

108. Dataclass example

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: int
    y: int

p = Point(2, 3)
print(p)
```

109. Type hints & annotations

```python
def add(x: int, y: int) -> int:
    return x + y

print(add(3, 4))
```

110. Lambda + map + filter

```python
nums = [1,2,3,4,5,6]
squares = list(map(lambda x: x**2, nums))
evens = list(filter(lambda x: x%2==0, nums))
print(squares, evens)
```

111. Reduce (functional programming)

```python
from functools import reduce
nums = [1,2,3,4,5]
product = reduce(lambda x,y: x*y, nums)
print(product)
```

112. JSON read & write

```python
import json
data = {"name":"Krish", "age":21}
with open("data.json", "w") as f:
    json.dump(data, f)

with open("data.json", "r") as f:
    print(json.load(f))
```

113. Random password generator

```
import random, string
chars = string.ascii_letters + string.digits + string.punctuation
password = "".join(random.choice(chars) for _ in range(12))
print(password)
```

114. Virtual environment info (sys module)

```
import sys
print(sys.prefix)   # shows current environment path
```

115. Unit testing (unittest)

```
import unittest

def add(x, y): return x+y

class TestMath(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2,3), 5)

# if __name__ == "__main__":
#     unittest.main()
```

116. Command-line arguments (sys.argv)

```
import sys
print("Arguments:", sys.argv)
```

117. Zip and enumerate

```
names = ["Alice","Bob","Charlie"]
scores = [85,90,88]
for name, score in zip(names, scores):
    print(name, score)

for idx, name in enumerate(names):
    print(idx, name)
```

118. Namedtuple

```python
from collections import namedtuple
Point = namedtuple("Point", "x y")
p = Point(3, 4)
print(p.x, p.y)
```

119. Counter (word frequency)

```python
from collections import Counter
words = "apple orange apple banana orange apple".split()
print(Counter(words))
```

120. Heapq (priority queue)

```python
import heapq
nums = [5,1,8,3]
heapq.heapify(nums)
heapq.heappush(nums, 0)
print(heapq.heappop(nums))
print(nums)
```