CodeBash · codebash.co.uk

FREE RESOURCE

20 Python Programming Challenges - Sample Solutions Pack

Multiple solution approaches for each challenge, including manual implementations that do not rely on built-in functions. For teacher use only.

Created by Eoin Shannon

CodeBash

Get all 100 challenges (with solutions pack) free at **codebash.co.uk/resources**

How to Use This Pack

This document contains solution notes and example code for all 100 challenges. Multiple approaches are provided where relevant - a built-in approach, a manual approach (without built-in functions), and a recursive or advanced approach where applicable. These are for teacher reference only.

Which Tier for Which Class?

Year Group	Recommended Tiers
KS3 (Year 7-9)	Tier 1 (Foundation), selected Tier 2 (Intermediate)
Year 10-11 (Secondary)	Tiers 1-3 (Foundation, Intermediate, Applied)
A-Level / Further	Tiers 3-4 (Applied, Stretch Challenges)

Foundation: Basic syntax, input/output, selection, and loops. Suitable for KS3 and secondary foundation learners.

Intermediate: Functions, lists, strings, OOP basics, and classic algorithms. Core secondary and lower A-Level.

Applied: File I/O, error handling, data structures, recursion, and applied algorithms. Upper secondary and A-Level.

Stretch Challenges: Multi-part problems combining multiple skills. A-Level standard. Maps to NEA complexity requirements.

Foundation

Basic syntax, input/output, selection, and loops. Suitable for KS3 and secondary foundation learners.

Challenge 5 Temperature Converter

PROBLEM

Write a program that converts a temperature from Celsius to Fahrenheit. Formula: F = (C x 9/5) + 32

EXAMPLE

```
Enter temperature in Celsius: 100
100 degrees C is 212.0 degrees F
```

MARK SCHEME

Correct formula. Output shows both values with units. Handles decimal input.

EXTENSION

Let the user choose the direction of conversion (C to F or F to C).

TEACHER NOTES

> Good for practising arithmetic and rounding. Students who finish quickly benefit from the extension, which introduces simple selection.

SOLUTIONS

Celsius to Fahrenheit

```python
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = (celsius * 9 / 5) + 32
print(f"{celsius}°C is {round(fahrenheit, 1)}°F")
```

Both directions (extension)

```python
direction = input("Convert C→F or F→C? ").strip().upper()
value = float(input("Enter temperature: "))
if direction == "C→F" or direction == "C>F":
    result = (value * 9 / 5) + 32
    print(f"{value}°C = {round(result, 1)}°F")
else:
    result = (value - 32) * 5 / 9
    print(f"{value}°F = {round(result, 1)}°C")
```

Challenge 8 Number Guessing Game

PROBLEM

Generate a random number between 1 and 10. Ask the user to guess it. Tell them if they are correct, too high, or too low. Keep asking until they get it right.

EXAMPLE

```
Guess a number between 1 and 10: 5
Too low!
Guess a number between 1 and 10: 8
Too high!
Guess a number between 1 and 10: 7
Correct! You took 3 guesses.
```

MARK SCHEME

Random number generated. Loop continues until correct. High/low feedback given. Guess count displayed.

EXTENSION

Give the user a maximum of 5 guesses before revealing the answer.

TEACHER NOTES

> This task works well once students have covered loops and selection. Some students struggle to understand that the random number must be generated before the loop begins, not inside it.

SOLUTIONS

Standard approach

```python
import random
secret  = random.randint(1, 10)
guesses = 1
guess   = int(input("Guess a number between 1 and 10: "))
while guess != secret:
    if guess < secret:
        print("Too low!")
    else:
        print("Too high!")
    guess = int(input("Guess a number between 1 and 10: "))
    guesses += 1
print(f"Correct! You took {guesses} guesses.")
```

With maximum guess limit (extension)

```python
import random
secret  = random.randint(1, 10)
guesses = 0
limit   = 5
while guesses < limit:
    guess = int(input(f"Guess (attempt {guesses + 1}/{limit}): "))
    guesses += 1
    if guess < secret:
        print("Too low!")
    elif guess > secret:
        print("Too high!")
    else:
        print(f"Correct in {guesses} guesses!")
        break
else:
    print(f"Out of guesses! The answer was {secret}.")
```

Challenge 9 FizzBuzz

PROBLEM

Print the numbers from 1 to 100. For multiples of 3 print Fizz, for multiples of 5 print Buzz, for multiples of both print FizzBuzz.

EXAMPLE

```
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz...
```

MARK SCHEME

Correct output for all three cases. FizzBuzz case handled before the individual checks.

EXTENSION

Make the range and divisors configurable by user input.

TEACHER NOTES

> FizzBuzz is deliberately simple but tests whether students understand order of conditions in if/elif chains. Students who check 3 and 5 separately first will fail on 15 - worth discussing why as a class.

SOLUTIONS

Standard approach

```python
for i in range(1, 101):
    if i % 15 == 0:
        print("FizzBuzz")
    elif i % 3 == 0:
        print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)
```

Configurable divisors (extension)

```python
n    = int(input("Count up to: "))
div1 = int(input("First divisor: "))
div2 = int(input("Second divisor: "))
w1   = input("Word for first: ")
w2   = input("Word for second: ")
for i in range(1, n + 1):
    if i % (div1 * div2) == 0:
        print(w1 + w2)
    elif i % div1 == 0:
        print(w1)
    elif i % div2 == 0:
        print(w2)
    else:
        print(i)
```

Challenge 17 Palindrome Checker

PROBLEM

Ask the user for a word. Tell them whether it is a palindrome (reads the same forwards and backwards).

EXAMPLE

```
Enter a word: racecar
racecar is a palindrome.
```

MARK SCHEME

Correct result for palindromes and non-palindromes. Case-insensitive comparison.

EXTENSION

Handle phrases (remove spaces and punctuation before checking). 'A man a plan a canal Panama' should return palindrome.

TEACHER NOTES

> Introduces string slicing. Students who check individual characters in a loop also get a working solution - praise the approach and then show how slice notation compresses it.

SOLUTIONS

Standard approach

```python
word = input("Enter a word: ").lower()
if word == word[::-1]:
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Manual reversal (no slice)

```python
word = input("Enter a word: ").lower()
reversed_word = ""
for char in word:
    reversed_word = char + reversed_word
if word == reversed_word:
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Phrase palindrome — ignores spaces/punctuation (extension)

```python
import string
phrase = input("Enter a phrase: ").lower()
cleaned = ""
for char in phrase:
    if char in string.ascii_lowercase:
        cleaned += char
if cleaned == cleaned[::-1]:
    print("Palindrome!")
else:
    print("Not a palindrome.")
```

Challenge 21

Leap Year Checker

PROBLEM

Ask the user for a year. Tell them whether it is a leap year. Rules: divisible by 4, except centuries, unless divisible by 400.

EXAMPLE

```
Enter a year: 2000
2000 is a leap year.
```

MARK SCHEME

Correct result for all cases: 2000=leap, 1900=not, 2024=leap, 2023=not.

EXTENSION

Display all leap years between two years entered by the user.

TEACHER NOTES

> A good test of nested or compound conditions. Test students with 1900 and 2000 specifically - those are the edge cases that catch out most solutions.

SOLUTIONS

Standard approach

```python
year = int(input("Enter a year: "))
if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

List all leap years in a range (extension)

```python
start = int(input("Start year: "))
end   = int(input("End year: "))
for year in range(start, end + 1):
    if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):
        print(year)
```

Intermediate

Functions, lists, strings, OOP basics, and classic algorithms. Core secondary and lower A-Level.

Challenge 29 Caesar Cipher

PROBLEM

Encrypt a message using the Caesar cipher. Ask for the message and the shift value.

EXAMPLE

```
Message: hello
Shift: 3
Encrypted: khoor
```

MARK SCHEME

Correct encryption for all lowercase letters. Non-letter characters unchanged. Wrap-around handled.

EXTENSION

Add a decryption function. Allow uppercase letters.

TEACHER NOTES

> Links well with theory lessons on encryption. The wrap-around is the tricky part - students who get most letters right but fail on x, y, z have probably missed the modulo.

SOLUTIONS

Lowercase only

```python
def encrypt(message, shift):
    result = ""
    for char in message:
        if char.isalpha() and char.islower():
            result += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
        else:
            result += char
    return result

message = input("Message: ")
shift   = int(input("Shift: "))
print("Encrypted:", encrypt(message, shift))
```

With uppercase + decryption (extension)

```python
def caesar(text, shift, decrypt=False):
    if decrypt:
        shift = -shift
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

message = input("Message: ")
shift   = int(input("Shift: "))
enc     = caesar(message, shift)
dec     = caesar(enc, shift, decrypt=True)
print(f"Encrypted: {enc}")
print(f"Decrypted: {dec}")
```

Challenge 33

Fibonacci Sequence

PROBLEM

Write a function that returns the first n Fibonacci numbers as a list.

EXAMPLE

```
How many? 8
[0, 1, 1, 2, 3, 5, 8, 13]
```

MARK SCHEME

Correct sequence. Function returns a list. Handles n=1 and n=2 correctly.

EXTENSION

Write a recursive version. Compare outputs with the iterative version.

TEACHER NOTES

> Check that students handle n=1 (returns [0]) and n=2 (returns [0, 1]) as edge cases. The extension to recursion works well as a lead-in to later recursive challenges.

SOLUTIONS

Iterative

```python
def fibonacci(n):
    if n == 1:
        return [0]
    seq = [0, 1]
    for _ in range(n - 2):
        seq.append(seq[-1] + seq[-2])
    return seq

n = int(input("How many? "))
print(fibonacci(n))
```

Recursive (extension)

```python
def fib(n):
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)

n = int(input("How many? "))
print([fib(i) for i in range(n)])
```

Challenge 37

Stack Implementation

PROBLEM

Implement a stack using a list. Provide push, pop, peek, and is_empty operations as functions.

EXAMPLE

```
push(5), push(3), push(9)
peek() -> 9
pop() -> 9
peek() -> 3
```

MARK SCHEME

All four operations work correctly. pop() and peek() handle empty stack gracefully.

EXTENSION

Use your stack to check if brackets in a string are balanced, e.g. {[()]} is balanced.

TEACHER NOTES

> Best taught alongside theory content on stacks and queues. Students often understand the concept easily but struggle to connect it to code - the append/pop mapping is worth making explicit.

SOLUTIONS

Stack using a list

```python
def push(stack, value):
    stack.append(value)

def pop(stack):
    if is_empty(stack):
        return None
    return stack.pop()

def peek(stack):
    if is_empty(stack):
        return None
    return stack[-1]

def is_empty(stack):
    return len(stack) == 0

s = []
push(s, 5); push(s, 3); push(s, 9)
print("Peek:", peek(s))
print("Pop:", pop(s))
print("Peek:", peek(s))
```

Bracket checker using the stack (extension)

```
def is_balanced(text):
    stack    = []
    matching = {")": "(", "]": "[", "}": "{"}
    for char in text:
        if char in "([{":
            stack.append(char)
        elif char in ")]}":
            if not stack or stack[-1] != matching[char]:
                return False
            stack.pop()
    return len(stack) == 0

print(is_balanced("{[()]}"))  # True
print(is_balanced("{[(])}"))  # False
```

Challenge 44

Bank Account Class

PROBLEM

Create a BankAccount class with attributes for account holder name and balance. Add methods for deposit, withdraw, and display balance. Validate that withdrawals do not exceed the balance.

EXAMPLE

```
account = BankAccount('Alice', 500)
account.deposit(200)
account.withdraw(100)
account.display()  -> Balance: 600
```

MARK SCHEME

Class defined with __init__. Correct deposit/withdraw methods. Validation prevents negative balance.

EXTENSION

Add a transaction history list. Add a print_statement() method.

TEACHER NOTES

> A good first OOP task because the real-world analogy is strong. Students often confuse the class definition with creating an object - make sure they write and run the instantiation line in the same program.

SOLUTIONS

BankAccount class

```python
class BankAccount:
    def __init__(self, holder, balance=0):
        self.holder  = holder
        self.balance = balance
        self.history = []

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            self.history.append(f"+£{amount:.2f}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds.")
        else:
            self.balance -= amount
            self.history.append(f"-£{amount:.2f}")

    def display(self):
        print(f"{self.holder}: Balance £{self.balance:.2f}")

account = BankAccount("Alice", 500)
account.deposit(200)
account.withdraw(100)
account.display()
```

Challenge 47

Bubble Sort

PROBLEM

Implement bubble sort to sort a list of numbers in ascending order. Do not use Python's built-in sort.

EXAMPLE

```
Input: [64, 34, 25, 12, 22, 11, 90]
Output: [11, 12, 22, 25, 34, 64, 90]
```

MARK SCHEME

Correct implementation. Sorted in ascending order. Works for any length list.

EXTENSION

Add an optimisation flag to stop early if no swaps occurred in a pass.

TEACHER NOTES

> Students who understand the concept but write incorrect index logic benefit from tracing through one full pass manually before coding. The early-exit extension is excellent for demonstrating best-case complexity.

SOLUTIONS

Bubble sort — ascending

```python
def bubble_sort(lst):
    n = len(lst)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
    return lst

data = [64, 34, 25, 12, 22, 11, 90]
print(bubble_sort(data))
```

Optimised with early-exit flag

```python
def bubble_sort_optimised(lst):
    n = len(lst)
    for i in range(n - 1):
        swapped = False
        for j in range(n - 1 - i):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
                swapped = True
        if not swapped:
            break
    return lst
```

Challenge 54 Hangman

PROBLEM

Implement a text-based Hangman game. Choose a random word from a predefined list. Give the player 6 lives.

EXAMPLE

```
Word: _ _ _ _ _
Guess a letter: e
Word: _ e _ _ _
Lives remaining: 6
```

MARK SCHEME

Word displayed as underscores. Letters revealed on correct guess. Lives decremented on wrong guess. Win/lose detected.

EXTENSION

Display an ASCII art gallows that builds progressively as lives are lost.

TEACHER NOTES

> Students enjoy this task and it combines several skills naturally. Common issues include not checking for repeated guesses and not detecting the win condition once the last letter is revealed.

SOLUTIONS

Hangman — 6 lives

```python
import random
words  = ["python", "hangman", "keyboard", "monitor", "variable"]
word   = random.choice(words)
guessed = []
lives  = 6

while lives > 0:
    display = " ".join(c if c in guessed else "_" for c in word)
    print(f"Word: {display}  Lives: {lives}")
    if "_" not in display:
        print("You win!")
        break
    guess = input("Guess a letter: ").lower()
    if guess in guessed:
        print("Already guessed.")
    elif guess in word:
        guessed.append(guess)
        print("Correct!")
    else:
        guessed.append(guess)
        lives -= 1
        print("Wrong!")
else:
    print(f"Game over! The word was: {word}")
```

Applied

File I/O, error handling, data structures, recursion, and applied algorithms. Upper secondary and A-Level.

Challenge 59

Error-Handled Calculator

PROBLEM

Build a calculator that handles division by zero, invalid input, and unknown operations gracefully using try/except.

EXAMPLE

```
Enter first number: 10
Enter operator (+, -, *, /): /
Enter second number: 0
Error: Cannot divide by zero.
```

MARK SCHEME

All four operations work. Division by zero handled. Non-numeric input handled. Unknown operator handled.

EXTENSION

Add support for ** (power) and % (modulo).

TEACHER NOTES

> Good for introducing exception handling. Students often catch all errors with a bare except clause - discuss why catching specific exceptions is better practice.

SOLUTIONS

Calculator with full error handling

```python
again = "y"
while again.lower() == "y":
    try:
        a  = float(input("First number: "))
        op = input("Operator (+, -, *, /): ")
        b  = float(input("Second number: "))
        if op == "+":
            print(f"Result: {a + b}")
        elif op == "-":
            print(f"Result: {a - b}")
        elif op == "*":
            print(f"Result: {a * b}")
        elif op == "/":
            if b == 0:
                print("Error: Cannot divide by zero.")
            else:
                print(f"Result: {a / b}")
        else:
            print("Unknown operator.")
    except ValueError:
        print("Invalid input. Please enter numbers only.")
    again = input("Calculate again? (y/n): ")
```

Challenge 63

Linked List

PROBLEM

Implement a singly linked list with Node and LinkedList classes. Provide methods to append, prepend, delete by value, and display.

EXAMPLE

```
list.append(1), append(2), append(3)
list.display() -> 1 -> 2 -> 3
list.delete(2)
list.display() -> 1 -> 3
```

MARK SCHEME

Node class with value and next pointer. Correct append and prepend. Delete handles missing value. Display traverses correctly.

EXTENSION

Add a method to reverse the linked list in place.

TEACHER NOTES

> Teach this alongside theory on linked lists. Students benefit from drawing boxes and arrows on paper before coding. The delete method is the hardest part - the edge case of deleting the head node catches many students out.

SOLUTIONS

Singly linked list

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

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, value):
        new_node = Node(value)
        if not self.head:
            self.head = new_node; return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def delete(self, value):
        if not self.head: return
        if self.head.value == value:
            self.head = self.head.next; return
        current = self.head
        while current.next:
            if current.next.value == value:
                current.next = current.next.next; return
            current = current.next

    def display(self):
        parts   = []
        current = self.head
        while current:
            parts.append(str(current.value))
            current = current.next
        print(" → ".join(parts))

ll = LinkedList()
ll.append(1); ll.append(2); ll.append(3)
ll.display()   # 1 → 2 → 3
ll.delete(2)
ll.display()   # 1 → 3
```

Challenge 68 Tower of Hanoi

PROBLEM

Write a recursive function to solve the Tower of Hanoi puzzle for n discs. Display each move.

EXAMPLE

```
n=3:
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
...
7 moves total.
```

MARK SCHEME

Correct recursive implementation. All moves displayed correctly. Total moves = 2^n - 1.

EXTENSION

Count and display the total number of moves alongside the solution.

TEACHER NOTES

> One of the classic recursion tasks. Students find it hard to write but satisfying when it works. Start with n=2 and trace through on the board - the pattern becomes clear and the code almost writes itself.

SOLUTIONS

Tower of Hanoi — recursive

```python
def hanoi(n, source, target, spare):
    if n == 1:
        print(f"Move disc 1 from {source} to {target}")
        return 1
    count  = hanoi(n - 1, source, spare,  target)
    print(f"Move disc {n} from {source} to {target}")
    count += hanoi(n - 1, spare,  target, source)
    return count + 1

n     = int(input("Number of discs: "))
total = hanoi(n, 'A', 'C', 'B')
print(f"Total moves: {total}  (expected: {2**n - 1})")
```

Challenge 70 Vigenere Cipher

PROBLEM

Implement the Vigenere cipher. Ask for a message and a keyword. Encrypt and display the result.

EXAMPLE

```
Message: hello
Keyword: key
Encrypted: rijvs
```

MARK SCHEME

Correct encryption for all lowercase letters. Keyword repeats correctly. Non-letter characters unchanged.

EXTENSION

Add decryption. Demonstrate how frequency analysis can help crack it.

TEACHER NOTES

> Best done after challenge 29 (Caesar cipher). The key repeating correctly is the hardest part - students often forget to use modulo on the key index.

SOLUTIONS

Vigenère cipher — encrypt

```python
def vigenere(text, key, decrypt=False):
    result  = ""
    key     = key.lower()
    key_idx = 0
    for char in text.lower():
        if char.isalpha():
            shift = ord(key[key_idx % len(key)]) - ord('a')
            if decrypt:
                shift = -shift
            result  += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
            key_idx += 1
        else:
            result += char
    return result

message = input("Message: ")
key     = input("Keyword: ")
enc     = vigenere(message, key)
dec     = vigenere(enc, key, decrypt=True)
print(f"Encrypted: {enc}")
print(f"Decrypted: {dec}")
```

Stretch Challenges

Multi-part problems combining multiple skills. A-Level standard. Maps to NEA complexity requirements.

Challenge 85

Noughts and Crosses

PROBLEM

Implement a two-player Noughts and Crosses game. Display the board after each move. Detect wins, draws, and invalid moves.

EXAMPLE

```
 X | O | X
---|---|---
 O | X |
---|---|---
   |   | O
X wins!
```

MARK SCHEME

Board displayed correctly after each move. All wins detected. Draw detected. Invalid moves rejected.

EXTENSION

Add a computer player that makes random moves. Then try to make it play optimally.

TEACHER NOTES

> Students are often familiar with the game, which helps. The win detection logic requires checking 8 combinations - encourage students to write this as a separate function rather than embedding it in the main game loop.

SOLUTIONS

Noughts and Crosses

```python
def display(board):
    for i in range(3):
        print(" | ".join(board[i*3:(i+1)*3]))
        if i < 2: print("---------")

def check_winner(board, player):
    wins = [(0,1,2),(3,4,5),(6,7,8),
            (0,3,6),(1,4,7),(2,5,8),
            (0,4,8),(2,4,6)]
    return any(board[a]==board[b]==board[c]==player for a,b,c in wins)

board    = [str(i+1) for i in range(9)]
current  = "X"
for _ in range(9):
    display(board)
    move = input(f"{current}'s turn (1-9): ")
    if not move.isdigit() or int(move) < 1 or int(move) > 9:
        print("Invalid."); continue
    idx = int(move) - 1
    if board[idx] in "XO":
        print("Taken."); continue
    board[idx] = current
    if check_winner(board, current):
        display(board)
        print(f"{current} wins!"); break
    current = "O" if current == "X" else "X"
else:
    display(board)
    print("It's a draw!")
```

Challenge 87 Spell Checker

PROBLEM

Load a dictionary of correctly spelled words from a file. For each word in a user-entered sentence, flag any word not in the dictionary and suggest the three closest matches.

EXAMPLE

```
Input: 'the quikc brwon fox'
Unknown: quikc -> suggestions: quick, thick, quack
Unknown: brwon -> suggestions: brown, brow, crown
```

MARK SCHEME

Correctly flags unknown words. Three suggestions produced for each. Case-insensitive matching.

EXTENSION

Implement the full Levenshtein distance algorithm for more accurate suggestions.

TEACHER NOTES

> Provide a word list file for students to load. The edit distance algorithm is the intellectual core of this task - give stronger students time to discover it and weaker students a worked example to implement.

SOLUTIONS

Spell checker with edit distance

```
DICTIONARY = ["quick","brown","fox","jumps","over","the","lazy","dog",
              "thick","quack","brick","crown","trick","frown"]

def edit_distance(a, b):
    m, n  = len(a), len(b)
    dp    = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m+1): dp[i][0] = i
    for j in range(n+1): dp[0][j] = j
    for i in range(1,m+1):
        for j in range(1,n+1):
            if a[i-1] == b[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
    return dp[m][n]

sentence = input("Enter sentence: ").lower().split()
for word in sentence:
    if word not in DICTIONARY:
        scored = [(edit_distance(word, d), d) for d in DICTIONARY]
        scored.sort()
        suggestions = [s[1] for s in scored[:3]]
        print(f"Unknown: {word} → suggestions: {', '.join(suggestions)}")
```

Challenge 89 Cryptarithmetic Solver

PROBLEM

Solve the puzzle SEND + MORE = MONEY by assigning a unique digit (0-9) to each letter.

EXAMPLE

```
S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
9567 + 1085 = 10652
```

MARK SCHEME

Correct solution found. Leading digit constraint applied. Solution displayed clearly.

EXTENSION

Generalise to solve any cryptarithmetic puzzle entered by the user.

TEACHER NOTES

> A good introduction to brute-force search. The program may take a few seconds to run - this is a natural lead-in to discussing constraint propagation and why smarter search algorithms exist.

SOLUTIONS

Cryptarithmetic SEND + MORE = MONEY

```python
from itertools import permutations

letters = "SENDMORY"
for perm in permutations(range(10), len(letters)):
    S,E,N,D,M,O,R,Y = perm
    if S == 0 or M == 0:
        continue
    SEND  = S*1000 + E*100 + N*10 + D
    MORE  = M*1000 + O*100 + R*10 + E
    MONEY = M*10000 + O*1000 + N*100 + E*10 + Y
    if SEND + MORE == MONEY:
        print(f"S={S} E={E} N={N} D={D} M={M} O={O} R={R} Y={Y}")
        print(f"{SEND} + {MORE} = {MONEY}")
        break
```

Challenge 92

Supermarket Queue Simulation

PROBLEM

Simulate a supermarket checkout queue. Customers arrive at random intervals. Each takes a random service time (1-5 mins). Run for 60 minutes. Display average waiting time and peak queue length.

EXAMPLE

```
Simulation complete.
Customers served: 23
Average wait: 2.4 minutes
Peak queue length: 7
```

MARK SCHEME

Simulation runs for exactly 60 minutes. Randomised arrivals and service. Both statistics calculated correctly.

EXTENSION

Add a second checkout that opens when the queue exceeds 5 people. Compare wait times.

TEACHER NOTES

> An accessible introduction to simulation as a problem-solving technique. The extension is excellent and directly demonstrates why supermarkets open extra checkouts at busy times.

SOLUTIONS

Supermarket queue simulation

```python
import random

queue          = []
time           = 0
server_free_at = 0
total_wait     = 0
served         = 0
peak_length    = 0

for time in range(60):
    if random.random() < 0.5:
        queue.append(time)
    if server_free_at <= time and queue:
        arrival        = queue.pop(0)
        wait           = time - arrival
        total_wait    += wait
        served        += 1
        service_time   = random.randint(1, 5)
        server_free_at = time + service_time
    if len(queue) > peak_length:
        peak_length = len(queue)

print(f"Customers served:   {served}")
print(f"Average wait:       {total_wait/served:.1f} mins" if served else "None served")
print(f"Peak queue length:  {peak_length}")
```

Challenge 94

Social Network

PROBLEM

Build a social network as an undirected graph using a dictionary of sets. Implement: add user, add friendship, suggest friends (friends-of-friends not already connected), find shortest connection path, and identify the most connected user.

EXAMPLE

```
suggest_friends('Alice') -> ['Charlie', 'Dave']
connection_path('Alice', 'Eve') -> Alice -> Bob -> Eve
```

MARK SCHEME

Graph correctly maintained as bidirectional. Friend suggestions correct. BFS for shortest path. Most connected user identified.

EXTENSION

Detect cliques - groups of three or more users who are all friends with each other.

TEACHER NOTES

> One of the more open-ended tasks in this set. Students who implement BFS for the shortest path are doing A-Level standard work. The friend suggestion algorithm alone is achievable for strong GCSE students.

SOLUTIONS

Social network graph

```
network = {}

def add_user(name):
    if name not in network:
        network[name] = set()

def add_friendship(a, b):
    add_user(a); add_user(b)
    network[a].add(b)
    network[b].add(a)

def suggest_friends(name):
    friends  = network.get(name, set())
    suggestions = set()
    for friend in friends:
        for fof in network[friend]:
            if fof != name and fof not in friends:
                suggestions.add(fof)
    return list(suggestions)

def connection_path(start, end):
    visited = set()
    queue   = [[start]]
    while queue:
        path = queue.pop(0)
        node = path[-1]
        if node == end: return path
        if node not in visited:
            visited.add(node)
            for n in network.get(node, []):
                queue.append(path + [n])
    return None

add_friendship("Alice", "Bob")
add_friendship("Bob",   "Charlie")
add_friendship("Alice", "Dave")
print(suggest_friends("Alice"))
print(connection_path("Alice", "Charlie"))
```

CodeBash

The interactive coding platform for UK schools

codebash.co.uk