

OCR GCSE COMPUTER SCIENCE

J277/02 – Paper 2 Revision Pack

Computational Thinking, Algorithms and Programming

Based entirely on OCR J277/02-mark schemes: June 2022, June 2023, Practice 2023, June 2024

How to use this pack:

Before the session	During the session
Read each section and tick off topics you feel confident about.	Work through unticked topics and attempt the quick practice questions.
Write Python code for any algorithm from memory.	Focus on Examiner Tips and Common Mistakes before the exam.

Programming language note: All code examples in this pack are written in Python, which you have studied. The examiner accepts correct Python code. OCR Reference Language is also accepted but Python is perfectly fine.

1 How the Paper 2 Exam Works

Detail	Information
Paper	J277/02 – Computational Thinking, Algorithms and Programming
Duration	1 hour 30 minutes
Marks	80 marks total
Sections	Section A: short/medium questions Section B: extended programming task (up to 8 marks)
Programming	You may answer in Python, OCR Reference Language, or pseudocode – be consistent
No calculator	No calculator is permitted
What's tested	Algorithms, programming constructs, logic, computational thinking, problem solving

Recurring Topics Across All Four Papers

These topics have appeared in every or nearly every paper – they are the highest-priority revision areas:

Topic	2022	2023	Practice 23	2024	Priority
Selection & iteration (if/while/for)	Yes	Yes	Yes	Yes	CRITICAL
Trace tables / tracing code	Yes	Yes	Yes	Yes	CRITICAL
Sorting algorithms (bubble, insertion, merge)	Yes	Yes	Yes	Yes	CRITICAL
Searching algorithms (binary, linear)	Yes	Yes	Yes	Yes	CRITICAL
Logic gates & truth tables	Yes	Yes	Yes	Yes	CRITICAL
SQL queries	Yes	Yes	Yes	Yes	CRITICAL
Functions / procedures (definition & call)	Yes	Yes	Yes	Yes	CRITICAL
Validation & test data	Yes	Yes	Yes	Yes	CRITICAL
Data types	Yes	Yes	Yes	Yes	CRITICAL
File handling	Yes	Yes	Yes	Yes	HIGH
Decomposition & abstraction	Yes	Yes	Yes	Yes	HIGH
Translators (compiler/interpreter)	Yes	Yes	Yes	Yes	HIGH
IDE features	No	Yes	No	No	MEDIUM
2D arrays	No	Yes	Yes	No	MEDIUM
String manipulation	No	No	Yes	Yes	MEDIUM
Maintainability / code quality	Yes	No	No	Yes	MEDIUM

2 Programming Constructs

2.1 The Three Constructs – You MUST Know the Correct Names

Construct	Definition (mark scheme wording)	Python keyword(s)
Sequence	Instructions executed one after another in order	Any line of code executed top-to-bottom
Selection	A decision is made; different code runs depending on a condition	if / elif / else
Iteration	A block of code is repeated / a loop	for / while

Examiner Tip: The exam often shows a line of code and asks you to identify the construct. Use the exact words: sequence, selection, or iteration.

Common Mistake: Writing 'loop' instead of 'iteration' may not gain the mark – always write 'iteration'.

2.2 Selection – if / elif / else

Mark schemes accept Python if/elif/else directly. Key points:

- Condition must evaluate to True or False
- elif is used for additional conditions (not 'else if' in Python)
- else catches anything not matched by if/elif

Example from Practice 2023 – pay calculation (3 bands):

```
hourly_rate = float(input('Enter hourly rate: '))
if hourly_rate < 7.50:
    band = 'Low'
elif hourly_rate <= 12.00:
    band = 'Standard'
else:
    band = 'High'
```

Mark Scheme: For a 3-band if/elif/else, award marks for: correct conditions, correct band assignment in each branch, and using elif (not a second if).

2.3 Iteration – for and while loops

Loop type	Use when...	Python syntax
for loop	Number of iterations is KNOWN in advance	for i in range(n):
while loop	Loop until a CONDITION is met; iterations not known in advance	while condition:
Count-controlled	Same as for loop – runs a fixed number of times	for i in range(start, stop, step):
Condition-controlled	Same as while loop – runs until condition is false	while x != 'quit':

Examiner Tip: The June 2023 mark scheme explicitly asks you to JUSTIFY your choice of loop. Say 'I used a while loop because the number of iterations is not known in advance' (e.g., user could enter 'stop' at any point).

Common Mistake: Do not say 'while is better'. You must give a REASON linked to the problem.

Classic while loop with validation (from multiple papers):

```
score = int(input('Enter score: '))
while score < 0 or score > 100:
    print('Invalid. Enter a score between 0 and 100.')
    score = int(input('Enter score: '))
```

for loop with range (from June 2022):

```
total = 0
for i in range(10):
    num = int(input('Enter a number: '))
    total = total + num
average = total / 10
print(average)
```

2.4 Python Operators – Know All of These

Operator	Meaning	Python symbol	Example	Result
Addition	Add	+	5 + 3	8
Subtraction	Subtract	-	5 - 3	2
Multiplication	Multiply	*	5 * 3	15
Division (real)	Divide – gives float	/	7 / 2	3.5
Integer division (DIV)	Divide – whole number only	//	7 // 2	3
Modulus (MOD)	Remainder after division	%	7 % 2	1
Exponentiation / Power	Raise to power	**	2 ** 3	8
Equal to	Comparison	==	x == 5	True/False
Not equal to	Comparison	!=	x != 5	True/False
Greater than	>	>	x > 5	True/False
Greater or equal	>=	>=	x >= 5	True/False
Less than	<	<	x < 5	True/False
Less or equal	<=	<=	x <= 5	True/False

Common Mistake: MOD is % in Python (not MOD). DIV is // in Python (not DIV). The mark scheme accepts either OCR notation or Python – but be consistent.

Examiner Tip: Practice 2023 traces: x=7 MOD 3 gives 1; x = 14 DIV 3 gives 4. Make sure you can work these out mentally.

3 Data Types

3.1 Core Data Types – Know Definitions and Examples

Data Type	Definition	Example values	Python casting
Integer	Whole number (no decimal point)	5, -3, 0, 100	int(x)
Real / Float	Number with a decimal point / fractional part	3.14, -0.5, 2.0	float(x)
Boolean	Can only be True or False	True, False	bool(x)
String	A sequence of characters	"hello", "123", ""	str(x)
Char	A single character	"A", "5", "!"	N/A in Python

Examiner Tip: When asked to choose between Boolean and String for a yes/no value, say: 'Boolean – because it can only hold two values: True or False, which matches a yes/no scenario.' (June 2023 mark scheme exact answer)

Common Mistake: Real and Float are interchangeable terms – OCR accepts both. Never write 'decimal' as a data type name.

3.2 Casting (Type Conversion)

Casting means converting a value from one data type to another.

Conversion	Python code	Example	Result
String to Integer	int(x)	int("42")	42
String to Float	float(x)	float("3.14")	3.14
Integer to String	str(x)	str(42)	"42"
Float to Integer	int(x)	int(3.9)	3 (truncates, does NOT round)

Mark Scheme: The term for converting a data type is 'casting'. Mark schemes credit: 'casting', 'type conversion', or 'type casting'.

Why do you need casting? input() in Python ALWAYS returns a String. You must cast to Integer or Float before doing arithmetic.

```
age = int(input('Enter your age: ')) # cast String to Integer
height = float(input('Enter height: ')) # cast String to Float
```

4 Searching and Sorting Algorithms

4.1 Linear / Sequential Search

How it works: Start at the first element. Check each element one by one until the target is found or the list ends.

```
def linear_search(lst, target):
    for i in range(len(lst)):
        if lst[i] == target:
            return i      # return the index
    return -1            # not found
```

Aspect	Detail
Works on	Unsorted OR sorted lists
Best case	Target is first element – $O(1)$
Worst case	Target not found or last element – $O(n)$
Mark scheme phrase	"Starts at beginning and checks each item in turn"

4.2 Binary Search

CRITICAL: The list MUST be sorted before binary search can work.

How it works:

- Find the middle element of the list
- If middle = target, return it
- If target < middle, search the LEFT half
- If target > middle, search the RIGHT half
- Repeat until found or no elements remain

Middle index formula: $\text{middle} = (\text{low} + \text{high}) // 2$

June 2024 trace example (find 10 in [2,4,6,8,10,12,14]):

```
# low=0, high=6, middle=3 -> lst[3]=8
# 10 > 8, so low=4, high=6, middle=5 -> lst[5]=12
# 10 < 12, so low=4, high=4, middle=4 -> lst[4]=10 FOUND
```

Aspect	Detail
Requirement	List MUST be sorted (ascending)
Best case	Middle element IS the target – $O(1)$
Worst case	$O(\log n)$ – much faster than linear for large lists
Mark scheme phrase	"Finds middle, discards half the list, repeats"

Examiner Tip: The June 2024 trace table question has 7 items. Middle index = $(0+6)//2 = 3$. Show your working in trace tables.

Common Mistake: Do NOT say binary search is always better. It requires a sorted list. Linear search works on unsorted data.

4.3 Bubble Sort

How it works: Repeatedly compare adjacent pairs. Swap if they are in the wrong order. Repeat passes until no swaps occur in a full pass.

```
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
```

June 2024 gap-fill key answers:

- Outer loop: `range(theTeam.length()-1)` → in Python: `range(len(theTeam)-1)`
- Swap variable name: count (used as loop counter for number of passes)
- Temp variable for swap: studentName
- Swap flag: set to True when a swap occurs

Examiner Tip: Bubble sort is inefficient on large lists but simple to explain. Exam questions often ask you to complete a gap-fill of bubble sort code – know the structure perfectly.

4.4 Insertion Sort

How it works: Maintain a 'sorted' portion on the left. Take the next item from the unsorted portion and INSERT it into the correct position in the sorted portion.

Key vocabulary from mark schemes:

- 'Sorted partition' – the part already in order (grows from left)
- 'Unsorted partition' – the part not yet sorted (shrinks from left)
- 'Temp variable' – stores the item being inserted while shifting elements

```
def insertion_sort(lst):
    for i in range(1, len(lst)):
        temp = lst[i]          # item to insert
        j = i - 1
        while j >= 0 and lst[j] > temp:
            lst[j + 1] = lst[j]  # shift right
            j -= 1
        lst[j + 1] = temp        # insert in correct place
    return lst
```

Question type	Expected answer
What does temp store?	The value being moved/inserted; allows elements to shift without losing it

Question type	Expected answer
Sorted vs unsorted partition	Sorted: items at index 0 to i-1. Unsorted: items at index i to end.
Difference from bubble sort	Insertion sort builds a sorted section; bubble sort repeatedly swaps adjacent pairs
Similarity to bubble sort	Both are comparison-based; both have $O(n^2)$ worst case; both sort in-place

4.5 Merge Sort

Merge sort uses a divide and conquer strategy.

Steps:

- **DIVIDE:** repeatedly split the list in half until you have single elements
- **CONQUER:** single elements are trivially sorted
- **MERGE:** combine pairs of sorted sub-lists into larger sorted lists

Aspect	Detail
Strategy	Divide and conquer
Time complexity	$O(n \log n)$ – more efficient than bubble/insertion on large data
Mark scheme phrase	"Split list in half repeatedly, then merge back in sorted order"
When asked to name the strategy	Write 'divide and conquer' – this exact phrase

Examiner Tip: Practice 2023 specifically tests merge sort and divide and conquer. Know these three words: divide, conquer, merge.

5 Logic Gates and Truth Tables

5.1 The Three Basic Gates

Gate	Symbol shape	Boolean expression	Rule
AND	D-shape	$A \text{ AND } B$ or $A \cdot B$	Output is 1 ONLY if ALL inputs are 1
OR	Curved arrow shape	$A \text{ OR } B$ or $A + B$	Output is 1 if AT LEAST ONE input is 1
NOT	Triangle with bubble	NOT A or A'	Inverts the input: $0 \rightarrow 1$, $1 \rightarrow 0$

5.2 Truth Tables – Must Know All Combinations

AND gate truth table:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate truth table:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate truth table:

A	NOT A
0	1
1	0

5.3 Combined Circuits – Work Step by Step

June 2024: NOT A AND (B OR C) – complete the truth table

Strategy: add intermediate columns for each gate output.

A	B	C	NOT A	B OR C	NOT A AND (B OR C)
0	0	0	1	0	0
0	0	1	1	1	1

A	B	C	NOT A	B OR C	NOT A AND (B OR C)
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	1	0

Examiner Tip: Always work left to right through the expression. Add intermediate columns even if not asked – it helps you avoid mistakes. Check: NOT A means flip all A values first.

Common Mistake: A very common error: writing 1 for NOT 1. Remember NOT 1 = 0, NOT 0 = 1.

5.4 Reading Logic Circuit Diagrams

In circuit diagrams, trace the signal flow from inputs to output:

- Identify each gate type from its shape
- Label intermediate wires with expressions
- Write the final Boolean expression for the output

Example (June 2022): The circuit has inputs A and B, with B going through NOT first, then two AND gates. This gives: (NOT B) AND A AND (some other input).

Mark Scheme: For drawing/reading circuits: each gate type must be identified correctly. NOT before AND means the NOT output feeds INTO the AND gate.

6 Flowcharts

6.1 Flowchart Symbols – Must Know All Five

Symbol shape	Name	Used for	Example content
Oval / rounded rectangle	Terminal / Start / End	Beginning and end of a flowchart	START, END, STOP
Rectangle	Process	Any calculation or assignment	total = total + num
Parallelogram (slanted rectangle)	Input / Output	Getting data from user or displaying results	INPUT name, OUTPUT score
Diamond	Decision	A condition that is either True or False	score > 50?
Arrow	Flow line	Shows direction of flow between symbols	(connecting arrows)

Examiner Tip: June 2024 asks you to complete a flowchart for even/odd. The decision diamond must have exactly two exits labelled TRUE/FALSE (or YES/NO). Both branches must reconnect or terminate.

Common Mistake: Do NOT use a rectangle for input/output. The parallelogram is essential. Marks are lost for wrong shapes.

6.2 Flowchart Best Practice

- All decision diamonds must have exactly two labelled exits (True/False or Yes/No)
- Loops in flowcharts: the decision at the bottom (or top) of the loop must reconnect via an arrow
- All paths must lead to the END terminal – no 'dead ends'
- Inputs come BEFORE processing; outputs come AFTER

7 Functions and Procedures

7.1 Key Vocabulary

Term	Definition	Python keyword
Function	A named block of code that returns a value	def ... return
Procedure	A named block of code that does NOT return a value	def ... (no return)
Parameter	A variable in the function definition that receives a value when called	def func(param):
Argument	The actual value passed to a function when it is called	func(argument)
Return value	The value a function sends back to where it was called	return x
Calling a function	Using the function's name with arguments to execute it	result = func(5)

7.2 Writing Functions in Python

Structure of a function (from June 2024 – robot arm movement):

```
def move_arm(position, direction):
    if direction == 'left':
        position = position - 5
    elif direction == 'right':
        position = position + 5
    if position < 1:
        position = 1
    elif position > 512:
        position = 512
    return position
```

Calling the function:

```
new_position = move_arm(current_pos, 'left')
```

Function with MOD (from Practice 2023):

```
def calculate(x, y):
    result = x % y    # MOD in Python is %
    return result
```

```
answer = calculate(10, 3)    # calling the function, answer = 1
```

Examiner Tip: Mark schemes award separate marks for: (1) correct def with function name and parameters, (2) correct logic/body, (3) correct return statement, (4) correct function call with arguments.

Common Mistake: Forgetting 'return' in a function means it returns None. Always include return if a value is needed.

7.3 Procedures vs Functions – When to Use Each

File handling, output-only tasks, and setup routines are often procedures (no return). Calculations, searches, and validation checks that need to give a result back are functions (use return).

Example procedure (from June 2023 – file handling):

```
def write_to_file(filename, data1, data2):  
    myFile = open(filename, 'w')  
    myFile.write(data1)  
    myFile.write(data2)  
    myFile.close()
```

8 File Handling

8.1 File Handling in Python – Four Steps

Step	Purpose	Python code
1. Open	Open the file and specify the mode	<code>myFile = open('data.txt', 'r')</code>
2. Read or Write	Get data from or put data into the file	<code>data = myFile.read()</code> or <code>myFile.write(text)</code>
3. Close	Release the file so other programs can use it	<code>myFile.close()</code>
4. (Optional) Use 'with'	Automatically closes file even if an error occurs	<code>with open('data.txt', 'r') as myFile:</code>

Mode	Meaning
'r'	Read mode – opens existing file to read from it
'w'	Write mode – creates new file (or overwrites existing) to write to it
'a'	Append mode – adds to end of existing file without overwriting

Full file write procedure (from June 2023 – 6-mark question):

```
def save_data(filename, name, score):
    myFile = open(filename, 'w')
    myFile.write(name)
    myFile.write(score)
    myFile.close()
```

Examiner Tip: For a 6-mark file handling question, the marks are typically: define procedure with parameters (1), open file with correct mode (1), write first value (1), write second value (1), close file (1), and a spare/overall mark.

Common Mistake: Forgetting `myFile.close()` loses a mark. Forgetting the mode (e.g., 'w') in `open()` also loses a mark.

9 Validation and Testing

9.1 Types of Validation

Validation type	What it checks	Example
Range check	Value is within acceptable min/max limits	Score must be between 0 and 100
Type check	Data is of the correct data type	Name must be a String; age must be an Integer
Presence check	Field is not empty / has been filled in	Username cannot be blank
Length check	Input has correct number of characters	Password must be 8–16 characters
Format check	Data matches a required pattern/format	Date must be DD/MM/YYYY
Lookup check	Value exists in an accepted list	Country code must be in the approved list

Examiner Tip: June 2023 asks for TWO validation methods for a game score with justification. The most common acceptable answers are range check ('score must be between 1 and 10') and type check ('score must be an integer').

9.2 Writing Validation Code in Python

Validation loop – keep asking until valid input is entered:

```
score = int(input('Enter score (1-10): '))
while score < 1 or score > 10:
    print('Invalid score. Try again.')
    score = int(input('Enter score (1-10): '))
```

Combined validation (from Practice 2023 – drone pilot codes):

```
code = input('Enter code: ')
while len(code) != 6 or not code.isalpha():
    print('Code must be 6 letters.')
    code = input('Enter code: ')
```

Mark Scheme: For a 4-mark validation question: (1) input statement, (2) correct condition using AND/OR, (3) error message output, (4) repeat input inside loop.

9.3 Test Data – Three Types You Must Know

Test data type	Definition	Example (score 40-180)	Expected result
Normal / valid	Typical data within the accepted range	100	Accepted: 'Valid'
Boundary	Values exactly at the edge of the range	40, 180, 39, 181	40 and 180: Valid; 39 and 181: Invalid

Test data type	Definition	Example (score 40-180)	Expected result
Erroneous / invalid	Data outside the range OR wrong type entirely	200, -5, 'hello'	Rejected: 'Not valid'

Examiner Tip: Always give SPECIFIC values for test data, not just descriptions. '100' not 'a number in range'. '40 and 180' not 'numbers at the boundaries'.

Common Mistake: 'Abnormal data' is an acceptable alternative to 'erroneous data' in OCR mark schemes. Either term is credited.

9.4 Types of Errors

Error type	Definition	Example
Syntax error	Code breaks the rules of the programming language; program cannot run at all	misspelled keyword, missing bracket, wrong indentation, missing colon
Logic error	Program runs but produces incorrect results / does not do what was intended	using + instead of -, wrong condition (> instead of <)
Runtime error	Error that occurs while the program is running (crashes it)	dividing by zero, index out of range, accessing a file that does not exist

Examiner Tip: June 2024 asks for the DEFINITION of syntax error (2 marks) AND two examples (2 marks). Know both. The definition: 'code that breaks the rules/grammar of the programming language'.

Common Mistake: 'The program doesn't work' is NOT a definition of syntax error. You must say: 'breaks the rules of the language' or 'language cannot understand/parse the code'.

10 SQL Queries

10.1 SQL Structure – Three Essential Keywords

Every SQL query you will be asked to write uses this exact structure:

```
SELECT field1, field2
FROM TableName
WHERE condition
```

Keyword	Purpose	Notes
SELECT	Choose which fields/columns to display	Use * to select all fields
FROM	Specify which table to query	Table name must be exact
WHERE	Filter rows by a condition	Optional; leave out if no filtering needed

10.2 Examples from Past Papers

June 2022 – query customers from London with >5 orders:

```
SELECT CustomerName, OrderCount
FROM Customers
WHERE City = 'London' AND OrderCount > 5
```

Practice 2023 – query drone flights over 50,000 miles:

```
SELECT DroneID, Mileage
FROM FlightData
WHERE Mileage > 50000
```

June 2024 – query Year 11 teams:

```
SELECT TeamName
FROM TblResult
WHERE YearGroup = 11
```

Examiner Tip: SQL marks are typically: SELECT correct fields (1), correct table name after FROM (1), correct WHERE condition (1), AND used correctly if two conditions (1). Always write SELECT, FROM, WHERE on separate lines.

Common Mistake: Do NOT use quotes around numeric values in WHERE: WHERE YearGroup = 11 NOT WHERE YearGroup = '11'. Quotes are only for text/string values.

Operator in SQL	Meaning	Example
=	Equal to	WHERE City = 'London'
>	Greater than	WHERE Score > 50
>=	Greater than or equal to	WHERE Age >= 18
<	Less than	WHERE Price < 10
<=	Less than or equal to	WHERE Quantity <= 5

Operator in SQL	Meaning	Example
AND	Both conditions must be true	WHERE Age >= 18 AND City = 'London'

11 Strings and Arrays / Lists

11.1 String Manipulation in Python

Operation	Python code	Example	Result
Length	<code>len(s)</code>	<code>len('hello')</code>	5
Upper case	<code>s.upper()</code>	<code>'hello'.upper()</code>	'HELLO'
Lower case	<code>s.lower()</code>	<code>'HELLO'.lower()</code>	'hello'
Substring (slice)	<code>s[start:end]</code>	<code>'hello'[1:3]</code>	'el'
Single character	<code>s[index]</code>	<code>'hello'[0]</code>	'h'
Concatenation	<code>s1 + s2</code>	<code>'AB' + 'CD'</code>	'ABCD'
String from int	<code>str(n)</code>	<code>str(42)</code>	'42'
Find in string	<code>s.find(x)</code>	<code>'hello'.find('l')</code>	2 (index of first 'l')

June 2024 string manipulation example:

```
s = 'abcd1234'
print(s.upper())      # ABCD1234
print(s[0:4].lower()) # abcd (already lower)
# every other digit: s[5], s[7] -> '2', '4', '6', '8' using step
print(s[5] + s[6] + s[7] + s[8]) # not valid - only 8 chars (0-7)
# Correct: s = 'abcd1234', even-indexed digits
digits = ''
for i in range(4, 8, 2): # indices 4 and 6 -> '1' and '3'
    digits += s[i] * 2
```

Examiner Tip: Remember: Python strings are zero-indexed. `s[0]` is the FIRST character. `s[len(s)-1]` is the LAST.

Common Mistake: `s[1:4]` gives characters at index 1, 2, 3 (NOT index 4). The end value is exclusive.

11.2 Arrays / Lists in Python

Operation	Python code	Example
Declare a list	<code>name = [v1, v2, v3]</code>	<code>scores = [10, 7, 8, 5]</code>
Access element	<code>name[index]</code>	<code>scores[0]</code> gives 10
Change element	<code>name[index] = value</code>	<code>scores[0] = 12</code>
Length of list	<code>len(name)</code>	<code>len(scores)</code> gives 4
Add to end	<code>name.append(value)</code>	<code>scores.append(9)</code>
Max value	<code>max(name)</code>	<code>max(scores)</code> gives 12

Examiner Tip: June 2023 array question: arrays in OCR Reference Language are 1-indexed (start at 1), but Python lists are 0-indexed (start at 0). When converting OCR array code to Python, subtract 1 from all indices.

Common Mistake: A common error in June 2023: `total = total + scores[1]` was wrong because iteration started at index 0. Check your starting index carefully.

11.3 2D Arrays / Lists

A 2D array is a list of lists. Access with `[row][column]`:

```
grid = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

print(grid[0][0])    # 1 (row 0, column 0)
print(grid[1][2])    # 6 (row 1, column 2)
```

Searching a 2D array (from June 2023 – 6 marks):

```
def search_2d(grid, target):
    for row in range(len(grid)):
        for col in range(len(grid[row])):
            if grid[row][col] == target:
                return row, col
    return -1, -1    # not found
```

12 Computational Thinking

12.1 The Four Concepts

Concept	Definition	Example in context
Decomposition	Breaking a complex problem down into smaller, more manageable sub-problems	A quiz app broken into: input questions, check answers, calculate score, display result
Abstraction	Removing unnecessary detail to focus on the important/relevant information	A road map – shows roads but not every tree and building
Algorithmic thinking	Creating a step-by-step solution (algorithm) that can solve the problem	Writing a bubble sort algorithm to sort any list of numbers
Pattern recognition	Identifying similarities or patterns within or between problems	Recognising that sorting names uses the same logic as sorting numbers

Examiner Tip: June 2024 asks for an example of abstraction AND decomposition IN CONTEXT (relating to a given video platform scenario). Always link your answer to the specific scenario in the question, not a general definition.

Common Mistake: Do NOT just write the definition. The question usually asks for an example. Give a specific, relevant example from the scenario described.

June 2024 abstraction example (video platform): Removing details such as the server hardware specs, network routing details, or video encoding format – focusing only on user-relevant data like video title, duration, views.

June 2024 decomposition example: Splitting the platform into sub-problems: user login, video search, video playback, recommendations, comments.

13 High-Level Languages, Low-Level Languages, and Translators

13.1 High-Level vs Low-Level Languages

Aspect	High-Level Language (e.g., Python)	Low-Level Language (e.g., Assembly/Machine code)
Readability	Human-readable; uses English-like syntax	Difficult to read; uses mnemonics or binary
Portability	Can run on different hardware/OS with little change	Usually specific to one processor type
Abstraction	Hardware details are hidden from the programmer	Works directly with CPU registers and memory
Speed	Slower after translation	Executes directly; very fast
Translator needed?	Yes – compiler or interpreter required	Assembler (for assembly), or none (machine code)

13.2 Compiler vs Interpreter

Feature	Compiler	Interpreter
Translation	Translates the WHOLE program at once before running	Translates and runs ONE LINE AT A TIME
Output	Produces a separate executable file	No executable file produced
Errors	Reports ALL errors after compilation	Stops at the FIRST error found
Execution speed	Faster to run (already translated)	Slower to run (translates during execution)
Source code needed?	End user does NOT need source code	Source code is required each time
Run multiple times	Executable can be run many times without re-translating	Must re-translate each time it runs

Examiner Tip: June 2024 asks for TWO advantages of a compiler over an interpreter. Pick two from: executable file produced, faster execution, no source code needed by end user, runs multiple times without re-translation, shows all errors at once, can optimise code.

Common Mistake: 'It's faster' is too vague. Say: 'The compiled executable runs faster because it has already been translated into machine code.'

Low-level language advantages (from June 2024):

- Runs faster / more efficiently than translated high-level code
- Uses less memory (more control over hardware resources)
- Can access hardware directly (e.g., specific registers)

14 IDEs, Maintainability, and Defensive Design

14.1 IDE Features – June 2023 asked for 4 marks (2 features + descriptions)

Feature	What it does
Syntax highlighting / editor	Colours code by type (keywords, strings, variables) to make it easier to read
Error reporting / diagnostics	Highlights or underlines syntax errors as you type, before running
Debugger / stepping	Allows you to run code line-by-line (step through) to find logic errors
Variable watch / inspector	Displays the current values of variables while debugging
Breakpoints	Lets you pause execution at a specific line to inspect program state
Auto-completion / IntelliSense	Suggests code completions as you type
Run/execute button	Allows you to run the program from within the IDE without a separate terminal

Examiner Tip: For 2-mark IDE questions, give the feature NAME (1 mark) AND a clear description of what it does (1 mark). Just naming the feature without explaining it will only score 1.

14.2 Code Maintainability – June 2024 asked for 4 marks

Technique	How it improves maintainability	Example
Meaningful identifiers	Variable/function names describe their purpose; easier for others to understand	totalScore not ts; calculateAverage not calc
Comments	Explain what the code does; helps future programmers understand intent	# This loop adds up all the scores
Subroutines (functions/procedures)	Organise code into named blocks; can be reused; easier to test in isolation	def calculate_average(scores):
Constants	Named values that don't change; self-documenting; change in one place updates everywhere	MAX_SCORE = 100
Indentation	Makes structure clear; shows which code is inside loops/functions	Python enforces this; use consistently

Mark Scheme: June 2024 gives 2 marks per maintainability technique: 1 for identifying the technique, 1 for a specific example. Always give a code example.

14.3 Defensive Design

Defensive design means writing programs that anticipate errors and handle them gracefully.

Method	What it means
Input validation	Check that input meets required criteria before processing it
Authentication	Verify the identity of a user before allowing access (e.g., username + password)
Input sanitisation	Remove or escape potentially dangerous characters from user input (e.g., prevent SQL injection)
Anticipating misuse	Consider how users could accidentally or deliberately cause problems; handle these cases

Examiner Tip: June 2024: 'State one method of defensive design' – any ONE of the above earns the mark. If it asks to explain, also describe what it does.

15 Exam Technique – What Examiners Reward

15.1 General Rules

- Read the question carefully – identify exactly what is being asked (define, explain, state, write code, complete, identify, trace...)
- Answer in the context of the question – always refer back to the scenario given
- Use technical vocabulary – mark schemes reward precise terminology
- For 'give one example': give ONE clear, specific example – do not hedge with multiple vague answers
- For code questions: write syntactically correct Python – even minor errors may lose marks

15.2 Command Words Explained

Command word	What you must do
State / Name / Identify	Give a brief answer without explanation (usually 1 mark per item)
Define	Give the technical meaning of the term (must be precise)
Describe	Give details about what something is or how it works
Explain	Give a reason WHY or HOW, not just WHAT
Justify	Give reasons to support your choice, linking to the specific context
Complete (code/table/flowchart)	Fill in the missing parts – check every blank
Write an algorithm / code	Produce working code – marks for correct structure, logic, and syntax
Trace through	Follow the code step-by-step and fill in the trace table – show every change to every variable

15.3 Trace Table Technique

- Create a column for EVERY variable mentioned in the code
- Create a column for any output statements
- Work through the code ONE LINE AT A TIME
- Only write a new value in a cell when that variable changes
- Leave cells blank if the variable has not yet been assigned a value
- For loops: track the loop variable (i, count, etc.) changing each iteration

Examiner Tip: Practice 2023 trace: follow the logic of string building carefully. Start with empty string ". After each loop iteration, concatenate new characters. Show each stage.

15.4 6-Mark Algorithm Writing Questions

These questions appear every year in Section B and are worth 6 marks. The mark scheme typically awards marks for:

- 1 mark: Correct input(s)
- 1 mark: Loop structure (for/while with correct condition)
- 1 mark: Correct processing logic inside the loop

- 1 mark: Correct output or return value
- 1–2 marks: Overall working algorithm (correct structure + correct result)

June 2024 – 6-mark algorithm: input team names and scores, loop until 'stop', find highest score and team, output both.

```
highestScore = 0
highestTeam = ''
team = input('Enter team name (or stop): ')
while team != 'stop':
    score = int(input('Enter score: '))
    if score > highestScore:
        highestScore = score
        highestTeam = team
    team = input('Enter team name (or stop): ')
print('Highest score:', highestScore, 'by', highestTeam)
```

Examiner Tip: For 6-mark algorithm questions: write a complete Python solution. Even if your logic has a small error, you will still earn marks for correct structure (input, loop, condition, output).

Common Mistake: Do NOT use pseudocode if you know Python – the examiner credits Python fully. Using pseudocode inconsistently or mixing pseudocode with Python may lose marks.

15.5 Marks That Are Commonly Dropped

- Not closing a file after opening it
- Forgetting 'return' in a function
- Using = (assignment) instead of == (comparison) in a condition
- Starting array/list index at 1 instead of 0 in Python
- Not labelling TRUE/FALSE on flowchart decision branches
- Giving a vague description of an error type (e.g., 'program crashes') instead of the technical name
- Saying 'a number' as test data instead of giving a SPECIFIC value
- Not casting input() to int or float before arithmetic
- Writing 'else if' instead of 'elif' in Python
- Confusing compiler and interpreter – remember: compiler does everything at once; interpreter goes line by line

16 Quick Reference Sheet

Python Syntax Cheat Sheet

Task	Python code
Input (string)	<code>name = input('Enter name: ')</code>
Input (integer)	<code>age = int(input('Enter age: '))</code>
Input (float)	<code>height = float(input('Enter height: '))</code>
Print output	<code>print('Hello', name)</code>
if/elif/else	<code>if x > 10: ... elif x > 5: ... else: ...</code>
for loop (n times)	<code>for i in range(n):</code>
for loop (a to b)	<code>for i in range(a, b+1):</code>
while loop	<code>while condition: ...</code>
Define function	<code>def my_func(param): return value</code>
Call function	<code>result = my_func(argument)</code>
List	<code>myList = [1, 2, 3]</code>
List access	<code>myList[0] # first element</code>
List length	<code>len(myList)</code>
List append	<code>myList.append(value)</code>
String length	<code>len(myString)</code>
String upper	<code>myString.upper()</code>
String lower	<code>myString.lower()</code>
String slice	<code>myString[1:4] # chars at index 1,2,3</code>
Integer division	<code>7 // 2 # gives 3</code>
Modulus/remainder	<code>7 % 2 # gives 1</code>
Power	<code>2 ** 3 # gives 8</code>
Open file (write)	<code>f = open('file.txt', 'w')</code>
Write to file	<code>f.write(data)</code>
Close file	<code>f.close()</code>
SQL query template	<code>SELECT field FROM Table WHERE condition</code>

Key Definitions at a Glance

Term	One-line definition
Decomposition	Breaking a problem into smaller sub-problems
Abstraction	Removing unnecessary detail; focusing on what matters
Algorithm	A precise, step-by-step set of instructions to solve a problem

Term	One-line definition
Sequence	Instructions executed one after another in order
Selection	Code that makes a decision based on a condition (if/else)
Iteration	Repeating a block of code (a loop)
Validation	Checking that input data meets required criteria
Syntax error	Code that breaks the rules of the programming language
Logic error	Code that runs but produces incorrect results
Runtime error	Error that occurs while the program is running, causing it to crash
Casting	Converting a value from one data type to another
Compiler	Translates entire program to machine code before running it
Interpreter	Translates and executes code one line at a time
Procedure	Named block of code that does NOT return a value
Function	Named block of code that RETURNS a value
Parameter	Variable in a function definition that receives a passed value
Binary search	Search algorithm that halves the search space each time (sorted lists only)
Linear search	Search algorithm that checks each element in turn (any list)
Bubble sort	Sort by repeatedly swapping adjacent out-of-order elements
Insertion sort	Sort by inserting each element into its correct position in a growing sorted portion
Merge sort	Divide-and-conquer sort: split into halves, sort each, merge back

Good luck! You have prepared thoroughly. Trust your knowledge, read each question carefully, and show your working.