Tracing algorithms and trace tables

1.2.4 · Computational thinking · revision notes

Tracing means working through an algorithm by hand, line by line, to find its output or the value a variable holds at a given point. A trace table keeps track of every variable as it changes. Expect 'complete the trace table' and 'state the output' questions worth 1 to 6 marks, and questions asking what an algorithm does.

Key facts

- Trace table: records the value of each variable as an algorithm runs
- One column per variable, plus an output column
- Write a new value each time a variable changes
- Dry run: tracing an algorithm by hand, without running it on a computer
- range(a, b) gives a, a + 1, ... up to b − 1
- print(a, b) separates the values with a space
- Use a variable's current value on the right-hand side of an assignment
- A while condition is checked before every pass, including the final check that ends the loop

Notes

What a trace table is

- A **trace table** records the values of the variables, and any output, as you work through an algorithm by hand (a **dry run**).
- There is one column per variable, often a column for the output, and sometimes one for a condition (True/False).
- Each time a variable changes, write its new value in its column. Move down a row as you go, e.g. one row per pass through a loop.
- Tracing is used to find the output for given data, to find the value of a variable at a certain point, to work out what an algorithm does and to find logic errors.

How to trace

- Start with the values the variables have before the loop (the first row).
- Work through one line at a time. Use the **current** values on the right-hand side of each assignment.
- Check the condition every time you return to the top of a `while` loop. For a `for` loop, list the values the counter takes, e.g. `range(1, 4)` gives 1, 2, 3.
- Write outputs exactly as the program would show them. `print(a, b)` puts a space between the two values.

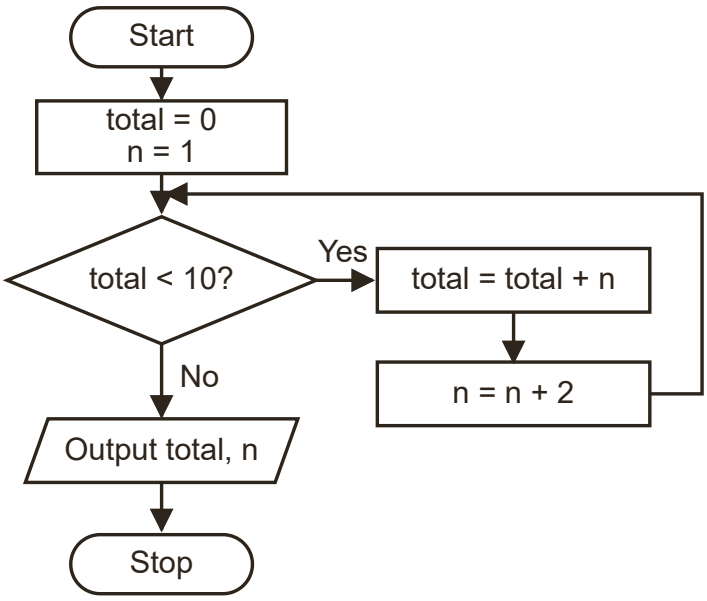
Worked example

- Trace this program.

```
total = 0
n = 1
while total < 10:
    total = total + n
    n = n + 2
print(total, n)
```

total	n	output
0	1	
1	3	
4	5	
9	7	
16	9	
		16 9

- The condition is checked before each pass: 0, 1, 4 and 9 are less than 10, so the loop runs four times. Then 16 < 10 is False, so the loop stops and the output is `16 9`.



The condition is checked before every pass. The loop body runs four times, then 16 < 10 is False

- Looking for a pattern tells you the purpose: this algorithm adds odd numbers (1 + 3 + 5 + 7) until the total reaches 10 or more. Grade 7+ only

How to answer each type of question

Complete a trace table

3 to 6 marks **Grade 5**

1. Fill in the first row with the starting values.
2. Go through the loop one pass at a time, writing each variable's new value.
3. At each if, check the condition with the current values before changing anything.
4. Finish with the output, written exactly as it would be displayed.

Example. Here is a program.

```
numbers = [4, 9, 2, 7]
biggest = numbers[0]
count = 0
for i in range(1, len(numbers)):
    if numbers[i] > biggest:
        biggest = numbers[i]
        count = count + 1
print(biggest, count)
```

Complete the trace table. The first row shows the values before the loop starts. You may not need all the rows.

i	biggest	count	output
	4	0	

Model answer.

i	biggest	count	output
	4	0	
1	9	1	
2	9	1	
3	9	1	
			9 1

i column: 1, 2, 3 (1); biggest column: 4, 9, 9, 9 (1); count column: 0, 1, 1, 1 (1); output: 9 1 (1).

Values written only when they change are also accepted.

State the output or the value of a variable

1 to 2 marks | Grade 4

1. Trace the algorithm, even if only in rough.
2. Stop at the point the question asks about.
3. Give the value exactly, with a decimal point if the result is a real number.

Example. Here is a program.

```
a = 7
b = 2
a = a + b
b = a * b
a = a - b
print(a, b)
```

State the output of this program.

Model answer.

a becomes 9, then b becomes 9 × 2 = 18 (1), then a becomes 9 − 18 = −9 (1).

Output: `-9 18`

State the purpose of an algorithm

1 to 3 marks | Grade 7

1. Trace the algorithm with the data given (or with simple data of your own).
2. Look at how the output is related to the input.
3. Describe what the algorithm does in one sentence, not line by line.

Example. Here is a program.

```
n = int(input("Enter a number: "))
result = 0
while n > 0:
    result = result + n % 10
    n = n // 10
print(result)
```

(a) State the output when the user enters 473.
(b) State the purpose of the algorithm.

Model answer.

(a) 14 (1): result goes 3, 10, 14 while n goes 47, 4, 0

(b) It adds up the digits of the number that is entered (1).

Don’t lose marks

- Using a variable's new value too early, before the line that changes it.

- Doing one pass too many or too few, e.g. running `range(1, 4)` with i = 4.
- Forgetting the final check of a while loop's condition, or carrying on after it is False.
- Writing a whole number where Python would show a real number: `8 / 2` is shown as 4.0.
- Leaving the output column until the end and then forgetting it.

More tips

Memory tricks

- Put a ruler or your finger on the current line and move it down as you trace.
- Tick each pass of a loop in the margin so that you do not lose count.
- Count the passes first: `for i in range(1, 4)` runs 3 times; a while loop runs until its condition is False.
- Sense check your final answer against what the algorithm seems to do.

Exam technique

- The table often has one row per pass. If it says 'you may not need all the rows', leave the extra rows blank.
- Each column often carries its own mark, so a slip in one column need not cost you the others. Keep going.
- Write output exactly as the screen would show it: `print("Hi")` shows Hi, without quotation marks.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3 — **Follow a short sequence to find the output**
Work through a few assignments and a print statement in order.

Grade 4 — **Trace a simple loop in a trace table**
Record each variable's new value on each pass through the loop.

Grade 5 — **Trace selection inside a loop**
Check the if condition on every pass and only update the variables it changes.

Grade 6 — **Trace algorithms that use arrays or strings**
Use the index on each pass to find the correct item or character.

Grade 7 — **State the purpose of an algorithm**
Spot the pattern in the trace, e.g. 'it adds up the digits of a number'.

Grade 8 — **Trace nested loops and 2D arrays accurately**
Keep track of both loop counters as the inner loop runs in full on each pass of the outer loop.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.