1.3 Iteration – doing it again

Name: _____ Class: _____ Date: _____ **Total: 22 marks**

KEY WORDS iteration • infinite loop • trace table

Objective

Choose `for` or `while` deliberately, trace loops by hand with a table, and guarantee termination. `break` and `continue` are used where they read best.

1 Key knowledge and worked examples

■ for when counted, while when conditioned

Iteration repeats work. A `for` loop walks a known sequence – `range(n)` counts `0..n-1`, and the loop variable takes each value in turn. A `while` loop runs as long as its condition holds.

```
total = 0
for i in range(1, 101):
    total += i
print(total)                    # 5050
```

Every `while` shows three things: the start value, the condition, and a step inside the body that moves toward exiting. Miss the step and you have an **infinite loop** .

```
n = 5
factorial = 1
while n > 1:
    factorial *= n
    n -= 1
print(factorial)
```

after	n	factorial	n > 1 ?
start	5	1	True
round 1	4	5	True
round 2	3	20	True
round 3	2	60	True
round 4	1	**120**	False

False: the loop ends and 120 is printed

■ Worked example: the trace table

The factorial loop, traced:

```
n = 5
factorial = 1
while n > 1:
    factorial *= n
    n -= 1
print(factorial)
```

round	n (start)	factorial (end)
1	5	5
2	4	20
3	3	60
4	2	120

Loop ends when `n` reaches 1; prints `120`. Build the **trace table** by hand, then put `print(n, factorial)` in the body and let Python grade you. The hand-run and the machine-run must agree – when they don't, you have found exactly where your mental model is wrong.

■ break and continue

`break` leaves the loop immediately; `continue` skips to the next round. Scanning until found reads best with `break`:

```
marks = [72, 85, 61, 90]
best_so_far = 0
for m in marks:
    if m > best_so_far:
        best_so_far = m
print(best_so_far)              # 90
```

Neither keyword is needed often – a well-chosen loop condition usually reads better than either.

2 Practice

2.1 Write a `for` loop that prints the 7-times table lines `7 x 1 = 7` up to `7 x 5 = 35`, using an f-string. [3]

2.2 Trace this loop with a table and give the printed result:

```
x = 100
steps = 0
while x > 1:
    x = x // 2
    steps += 1
print(steps)
```

[4]

2.3 State one change that would make the loop in 2.2 run forever, and why the original terminates. [3]

3 Exam-style questions

3.1 A savings model: a balance starts at 1000 and grows by 5% each year, rounded down to whole currency units. Write a loop that prints the year and balance each year, stopping as soon as the balance reaches 2000, and give the first year that happens. Then state which loop type you chose and why. [6]

3.2 A word game scores words: 1 point per letter, but a word containing "q" scores double. Given a list `words = ["cat", "quilt", "hi", "queue"]`, write a loop that prints each word with its score, and the total of all scores. Trace the total by hand. [6]

Solutions

2.1

- `for i in range(1, 6):`
- `print(f"7 x {i} = {7 * i}")`

2.2

- Table: (100→50, steps 1), (50→25, 2), (25→12, 3), (12→6, 4), (6→3, 5), (3→1, 6).
- Prints `6`.

2.3

- Replace `x = x // 2` with `x = x / 2`: the result is a float that halves forever without reaching 1 (it approaches 0 but `x > 1` stays true for many rounds and float underflow takes essentially forever).
- Any change that stops `x` decreasing (e.g. removing the halving line) loops forever.
- The original terminates because `// 2` makes an `int` at least halve strictly and reaches 1 in finitely many steps.

3.1

```
balance = 1000
year = 0
while balance < 2000:
    year += 1
    balance = balance * 105 // 100
    print(year, balance)
```

- Condition on the goal (`while balance < 2000`) with the year step visible.
- Growth with floor rounding.
- Trace: 1050, 1102, 1157, 1214, 1274, 1337, 1403, 1473, 1546, 1623, 1704, 1789, 1878, 1971, 2069 → first year over 2000 is year 15 (balance 2069).
- `while` because the trip count is not known in advance – the condition decides.

3.2

```
total = 0
for word in words:
    score = len(word) * (2 if "q" in word else 1)
    print(word, score)
    total += score
print("Total:", total)
```

- Loop over words with the per-word score.
- The “q” doubling expressed once.
- Scores: cat 3, quilt 10, hi 2, queue 10.
- Total 25.