1.7 Decomposition, testing and debugging

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

KEY WORDS decomposition • testing • debugging • boundary case

Objective

Decompose a problem into single-job functions, write boundary tests the moment a function exists, and debug by bisecting the gap between what you believe and what the code does.

1 Key knowledge and worked examples

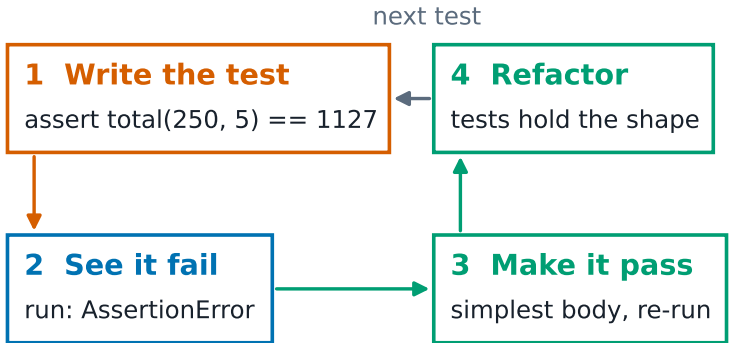
■ Decomposition: one job per function

Decomposition splits a problem until each piece does one thing – small enough to hold in your head, and to test alone. The functions from unit 1.4 compose:

```
def add_fee(amount):
    """Return amount plus a fixed 3 pence fee."""
    return amount + 3

def order_total(unit_pence, quantity):
    """Return the order total: subtotal plus the fixed fee."""
    return add_fee(unit_pence * quantity)
```

Each has a stated contract in its docstring: inputs, output, units. A function that prints instead of returning cannot be composed – printing is for the edges of the program, returning is for its middle.



■ Worked example: tests at the boundaries

Testing compares actual with expected. The boundary cases find the bugs: zero, one, empty, and one awkward case.

```
assert order_total(250, 4) == 1003   # a normal case
assert order_total(250, 0) == 3      # zero quantity - fee still applies
assert order_total(0, 4) == 3        # zero unit price
```

Silence is a pass: an `assert` that never fires is a green test. Write tests before the body when you can – the tests are the contract made executable, and a function written to pass them rarely needs its logic re-examined.

■ Debugging by bisection

Debugging is locating the difference between belief and behaviour. The method: (1) reproduce with the smallest failing input; (2) inspect values at the boundary between the code you trust and the code you suspect; (3) fix one thing; (4) re-run all the tests.

For a 100-line pipeline, one well-placed `print` halves the search each run – three runs locate a fault among eight stages. Remove the prints (or turn them into tests) when done.

2 Practice

2.1 Name the three boundary cases every new function should get, according to this course's habit, and apply them to `def repeat(text, times)` returning the text repeated `times` times. [3]

2.2 A function's tests all pass, but the program's output is still wrong. Give the two most likely locations of the fault, given the tests pass. [2]

2.3 Write the boundary tests for `def clip(value, low, high)` that returns `value` limited to the range `low..high`. [4]

3 Exam-style questions

3.1 A café report needs: the day's takings from a list of sale amounts, the count of sales, and a line `n sales, total t, average a` with the average at two decimals (0.00 when there are no sales).

Decompose this into functions of one job each (names, parameters, return values – docstrings optional), then write the boundary tests for the empty-sales case and one normal case. [7]

3.2 A student's `tally` function passes its tests, but the program printing `"Most popular: {winner}"` shows the wrong drink. The program reads orders from a file into a list, tallies with the function, then picks `max(counts, key=counts.get)`.

Set out a bisection debugging plan: the three inspection points you would add and what each would confirm, and name the most likely fault given the tests pass. [6]