ANSWERS

A2.3 Recursion versus iteration

Recursion and computational thinking · A level · OCR H446 2.2.1, AQA 7517 4.1.1.15, Eduqas A500QS 1.3 · about 20 min

What this lesson is about

The same algorithm both ways, the cost of a frame per call, repeated work, and stack overflow.

- The same algorithm, twice
- Turning recursion into a loop
- The cost of repeated work
- Stack overflow
- Comparing the two

Questions

5 marks in all

1. Why does a recursive solution usually use more memory than an iterative one? [1 mark]

- ☐ **A** Every call that has not yet returned holds a stack frame
- ☐ **B** Recursive code is longer
- ☐ **C** Loops store their variables on disk
- ☐ **D** Recursion copies the whole program each call

> **Answer:** A. Recursion depth n means n frames on the stack at once; a loop reuses one set of variables.

2. Which are advantages of recursion over iteration? [1 mark]

Tick every answer that is true.

- ☐ **A** It can express self-similar problems, such as tree traversal, more naturally
- ☐ **B** The code can be shorter and closer to the problem's definition
- ☐ **C** It uses less memory
- ☐ **D** It cannot cause a stack overflow

> **Answer:** A, B. Recursion's strengths are clarity for naturally recursive problems. It uses more memory and risks stack overflow.

3. What does this program print? [1 mark]

```
calls = 0

def fib(n):
    global calls
    calls = calls + 1
    if n < 2:
        return n
    return fib(n - 1) + fib(n - 2)

print(fib(5), calls)
```

Answer:

```
5 15
```

fib(5) is 5, but the two recursive calls repeat work: fib(3) is worked out twice and fib(2) three times, 15 calls in all.

4. What does this program print? [1 mark]

```
def total_loop(n):
    result = 0
    while n > 0:
        result = result + n
        n = n - 1
    return result

def total_rec(n):
    if n == 0:
        return 0
    return n + total_rec(n - 1)

print(total_loop(6), total_rec(6))
```

Answer:

```
21 21
```

Both add 6 + 5 + 4 + 3 + 2 + 1. The loop version keeps one result variable; the recursive one has seven frames at its deepest.

5. A recursive function to add up a list of 50,000 sensor readings crashes in Python, but a loop version works. Why? [1 mark]

- ☐ **A** The recursion is too deep: it needs about 50,000 frames, beyond the limit on the call stack
- ☐ **B** Recursion cannot add numbers
- ☐ **C** The loop version is compiled and the recursive one is not
- ☐ **D** The list is too large to store

Answer: A. One frame per item means a very deep call stack, which overflows. The loop needs no extra frames.

The task: binary, both ways

Write two functions that each take a whole number `n` from 0 to 1,000 and **return** its binary digits as a string, with no leading zeros (`0` gives `"0"`, `6` gives `"110"`): - `to_binary_rec(n)` must be recursive, with no loop; - `to_binary_loop(n)` must use a loop. Do not use `bin` or `format`. Then, for each `n` in `[0, 1, 6, 37, 255]`, print one line `<n>: <recursive answer> <loop answer>`, for example `6: 110 110`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def to_binary_rec(n):
    return ""

def to_binary_loop(n):
    return ""

for n in [0, 1, 6, 37, 255]:
    print(f"{n}: {to_binary_rec(n)} {to_binary_loop(n)}")
```

The hint students can ask for: The last binary digit of n is its remainder when divided by 2, and the other digits are the binary of n divided by 2 (whole-number division). For recursion, that is the general case; decide which small numbers are their own answer. For the loop, collect digits in a string, adding each new one at the front.

A solution

```
from bugbot import *
connect()

def to_binary_rec(n):
    if n < 2:
        return str(n)
    return to_binary_rec(n // 2) + str(n % 2)

def to_binary_loop(n):
    if n == 0:
        return "0"
    digits = ""
    while n > 0:
        digits = str(n % 2) + digits
        n = n // 2
    return digits

for n in [0, 1, 6, 37, 255]:
    print(f"{n}: {to_binary_rec(n)} {to_binary_loop(n)}")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.