ANSWERS

A2.2 Recursion

Recursion and computational thinking · A level · OCR H446 2.2.1, AQA 7517 4.1.1.16, Eduqas A500QS 1.3 · about 25 min

BugBotLab

What this lesson is about

Base case and general case, winding and unwinding, and a spiral the robot draws by calling itself.

- Base case and general case
- Winding and unwinding
- What goes wrong
- Recursion on the robot

Questions

6 marks in all

1. What is the base case of a recursive subroutine? [1 mark]

- ☐ **A** A case answered directly, with no further recursive call
- ☐ **B** The first call made from the main program
- ☐ **C** The case with the largest input
- ☐ **D** The line that calls the subroutine again

> **Answer:** A. The base case stops the recursion. The line that calls itself is the general case.

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

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

print(f(4))
```

> **Answer:**
>
> ```
> 10
> ```
>
> f(4) = 4 + 3 + 2 + 1 + 0 = 10, built up as the calls unwind.

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

```
def show(n):
    if n > 0:
        show(n - 1)
        print(n)

show(3)
```

Answer:

```
1
2
3
```

The print comes after the recursive call, so nothing is printed while winding. The numbers appear as the calls unwind, smallest first.

4. def count(n): return n * count(n - 1). What happens when count(3) is called? [1 mark]

- ☐ **A** It never reaches a base case and ends in a stack overflow (a RecursionError in Python)
- ☐ **B** It returns 6
- ☐ **C** It returns 0
- ☐ **D** It returns 3

Answer: A. There is no base case, so every call makes another, until the stack runs out.

5. factorial(n) returns 1 if n == 0, otherwise n * factorial(n - 1). How many calls of factorial are made in total when the main program calls factorial(4)? [1 mark]

Answer: 5. factorial(4), (3), (2), (1) and (0): five calls, and five frames at the deepest point.

6. Put the events of factorial(2) in the order they happen. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `factorial(1) returns 1`
___ `factorial(0) returns 1`
___ `factorial(1) calls factorial(0)`
___ `factorial(2) calls factorial(1)`
___ `factorial(2) returns 2`

Answer:

```
factorial(2) calls factorial(1)
factorial(1) calls factorial(0)
factorial(0) returns 1
factorial(1) returns 1
factorial(2) returns 2
```

All the calls wind down to the base case first; the returns then happen in reverse order.

The task: down and back up

Write a recursive procedure `echo(n)`, where `n` is a whole number from 0 upwards: - if `n` is 0 (the base case), print `base` and play `tone(200, 0.2)`; - otherwise print `down <n>` and play `tone(200 + 100 * n, 0.2)`, then call `echo(n - 1)`, then print `up <n>` and play `tone(200 + 100 * n, 0.2)` again. Call `echo(3)`. The output must be `down 3`, `down 2`, `down 1`, `base`, `up 1`, `up 2`, `up 3`, one per line, and you will hear the notes fall and rise again. Use no loops.

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

def echo(n):
    print("down", n)

echo(3)
```

The hint students can ask for: Decide what echo(0) does on its own, with no further call. For any bigger n, the work splits into three parts: something before the call on the smaller problem, the call itself, and something after it. The lines after the call only run once the smaller call has finished.

A solution

```
from bugbot import *
connect()

def echo(n):
    if n == 0:
        print("base")
        tone(200, 0.2)
    else:
        print("down", n)
        tone(200 + 100 * n, 0.2)
        echo(n - 1)
        print("up", n)
        tone(200 + 100 * n, 0.2)

echo(3)
```

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