ANSWERS

# A13.6 List processing

Functional programming · A level · AQA 7517 4.12.3.1, Eduqas A500QS 1.4 · about 25 min

BugBotLab

## What this lesson is about

Lists as a head and a tail, the empty list, prepend and append, and recursion over a list to drive a route there and back.

- Head and tail
- The list operations
- Recursion over a list
- Why prepend is cheap and append is not
- A route as a list

## Questions

**6 marks in all**

**1.** What is head [7,2,9]? [1 mark]

> **Answer:** 7. The head is the first element, and it is an element, not a list.

**2.** What is tail [7,2,9]? Write it as a Haskell list. [1 mark]

> **Answer:** [2,9]. The tail is the list of everything after the head.

**3.** What is tail [5]? [1 mark]

- ☐ **A** The empty list, []
- ☐ **B** 5
- ☐ **C** [5]
- ☐ **D** An error

> **Answer:** A. A one-item list is 5 : [], so its tail is the empty list. It is head [] and tail [] that are errors.

**4.** What is 3 : [8,1]? Write it as a Haskell list. [1 mark]

> **Answer:** [3,8,1]. The colon prepends an item to the front of a list.

**5.** What does this program print? [1 mark]

```
def f(xs):
    if xs == ():
        return ()
    if xs[0] > 10:
        return (xs[0],) + f(xs[1:])
    return f(xs[1:])

print(f((4, 15, 9, 22)))
```

> **Answer:**
>
> ```
> (15, 22)
> ```
>
> Each call keeps the head if it is over 10 and recurses on the tail, so this is filter written by recursion.

**6.** Which statements about lists in a functional language are true? [1 mark]

Tick every answer that is true.

- ☐ **A** The head of a list is an element
- ☐ **B** The tail of a list is a list
- ☐ **C** Taking the head of the empty list is an error
- ☐ **D** [1,2] ++ 3 appends 3 to the list

> **Answer:** A, B, C. ++ joins two lists, so appending one item needs it in a list of its own: [1,2] ++ [3].

## The task: there and back

Drive a route, then come back along it, using only list operations and recursion. There must be no `for` or `while`, no `len`, and no `reversed` or `[::-1]` anywhere. - Write `head(xs)`, `tail(xs)`, `is_empty(xs)`, `prepend(x, xs)` and `append(xs, x)` for tuples, as in the table above. - `length(xs)` returns the number of items in the tuple `xs`, by recursion. - `reverse(xs)` returns a new tuple with the items of `xs` in the opposite order, by recursion: the reverse of a list is the reverse of its tail, with its head appended. - `invert(move)` takes a move, a tuple `(letter, amount)` where the letter is `"F"`, `"B"`, `"L"` or `"R"` and the amount a whole number, and returns the move that undoes it: `"F"` and `"B"` swap, `"L"` and `"R"` swap, and the amount stays the same. - `drive(route)` carries out a tuple of moves by recursion. For each move, in order, it prints the letter and amount separated by a space, such as `F 30`, then drives forward or backward at speed 50 or turns left or right at speed 30. Using `ROUTE` from the starter, print `length: <n>` and `head: <the first move>` (Python's own printing of the tuple, such as `('F', 30)`). Then drive `ROUTE`, and then drive back to the start along the route undone: the inverted moves of the reversed route.

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

ROUTE = (("F", 30), ("R", 90), ("F", 20), ("L", 45), ("F", 10))

for move in ROUTE:
    print(move[0], move[1])
    forward(50, distance=move[1])
```

**The hint students can ask for:** Every recursive function here has the same shape: what is the answer for the empty tuple, and how is the answer for a whole tuple made from its head and the answer for its tail? To undo a journey, think about which move you must undo first.

### A solution

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

ROUTE = (("F", 30), ("R", 90), ("F", 20), ("L", 45), ("F", 10))

def head(xs): return xs[0]
def tail(xs): return xs[1:]
def is_empty(xs): return xs == ()
def prepend(x, xs): return (x,) + xs
def append(xs, x): return xs + (x,)

def length(xs):
    if is_empty(xs):
        return 0
    return 1 + length(tail(xs))

def reverse(xs):
    if is_empty(xs):
        return ()
    return append(reverse(tail(xs)), head(xs))

OPPOSITE = {"F": "B", "B": "F", "L": "R", "R": "L"}
```

```
def invert(move):
    return (OPPOSITE[move[0]], move[1])

COMMANDS = {"F": lambda n: forward(50, distance=n), "B": lambda n: backward(50,
distance=n),
            "L": lambda n: turn_left(30, angle=n), "R": lambda n: turn_right(30, angle=n)}

def drive(route):
    if is_empty(route):
        return
    letter, amount = head(route)
    print(letter, amount)
    COMMANDS[letter](amount)
    drive(tail(route))

print("length:", length(ROUTE))
print("head:", head(ROUTE))
drive(ROUTE)
drive(tuple(map(invert, reverse(ROUTE))))
```

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