WORKSHEET

A13.6 List processing

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

BugBotLab

Name _____ Class _____ Date _____

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]

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

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

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

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

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)))
```

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

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])
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a13-6-list-processing/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Write `last(xs)`, the final item of a list, using only `head`, `tail` and `is_empty`.
2. Write `take(n, xs)`, the first `n` items, in Haskell style with two equations, then in Python.
3. Rewrite `reverse` so it builds the answer by prepending, carrying a second parameter for the answer so far. Why would a Haskell programmer prefer it?