ANSWERS

A13.5 Map, filter and fold

Functional programming · A level · AQA 7517 4.12.2.1, Eduqas A500QS 1.4 · about 25 min

BugBotLab

What this lesson is about

The three higher-order functions, foldl and foldr, and a pipeline over a log of sensor readings.

- The log
- map: change every item
- filter: keep some items
- fold: combine everything into one value
- Putting them together
- On the robot

Questions

6 marks in all

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

```
cms = [12, 45, 30, 61]
print(list(map(lambda cm: cm * 10, filter(lambda cm: cm > 25, cms))))
```

Answer:

```
[450, 300, 610]
```

filter keeps 45, 30 and 61, in order; map then multiplies each by 10.

2. What does filter do? [1 mark]

- ☐ **A** Produces a new list containing exactly the items of the original that match a given condition
- ☐ **B** Applies a function to every item and returns the list of results
- ☐ **C** Reduces a list to a single value with a combining function
- ☐ **D** Removes the first item of a list

Answer: A. The second option describes map and the third fold. The original list is unchanged.

3. What is the value of foldl (-) 10 [1,2,3]? [1 mark]

Answer: 4. foldl starts at the left: ((10 - 1) - 2) - 3 = 4.

4. What is the value of foldr (-) 0 [5,3,1]? [1 mark]

Answer: 3. foldr starts at the right: 5 - (3 - (1 - 0)) = 5 - 2 = 3.

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

```
from functools import reduce

log = [(0, 40), (90, 15), (180, 62), (270, 33)]
far = reduce(lambda a, b: a if a[1] >= b[1] else b, log)
print(far[0])
```

Answer:

```
180
```

The combining function keeps the pair with the larger reading, so the fold ends with (180, 62) and prints its heading.

6. What is the result of the Haskell expression map (*2) [1,2,3]? Write it as a Haskell list. [1 mark]

Answer: [2,4,6]. map applies (*2), the function that doubles, to each item in turn.

The task: analyse the sweep

Analyse `sweep.csv` (above) with map, filter and fold. There must be no `for` or `while` anywhere in the program (so no comprehensions either), and no `min`, `max` or `sum`: use `map`, `filter` and `reduce` instead. - `parse(line)` takes one line of the file, such as `"2.4,60,24.5"`, and returns a tuple `(time, heading, cm)` with `time` a float, `heading` an int and `cm` a float. - Read the file, skip the header line, and map `parse` over the rest. - Keep only the **valid** readings, those under 400 cm, and print `valid: <how many>`. - Of the valid readings, print the headings of those over 40 cm, in the order they are in the file, as `clear: 0 30 ...` separated by spaces. - Fold the valid readings to find the nearest, and print `nearest: <cm> cm at <heading>`. - Fold the valid readings to their total, and print `mean: <mean cm>` rounded to one decimal place. The robot does not move.

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

f = open("sweep.csv", "r")
lines = f.read().splitlines()
f.close()

for line in lines[1:]:
    print(line)
```

The hint students can ask for: Work in stages, printing each one as you go: the parsed tuples, the valid ones, and so on. Each stage is a new value made from the one before. For the nearest, the combining function is given two readings and keeps one; for the total, it is given the total so far and a reading.

A solution

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

def parse(line):
    t, h, cm = line.split(",")
    return (float(t), int(h), float(cm))

f = open("sweep.csv", "r")
lines = f.read().splitlines()
f.close()

readings = tuple(map(parse, lines[1:]))
valid = tuple(filter(lambda r: r[2] < 400, readings))
print("valid:", len(valid))

clear = filter(lambda r: r[2] > 40, valid)
print("clear:", " ".join(map(lambda r: str(r[1]), clear)))

nearest = reduce(lambda a, b: a if a[2] <= b[2] else b, valid)
print("nearest:", nearest[2], "cm at", nearest[1])

total = reduce(lambda so_far, r: so_far + r[2], valid, 0)
print("mean:", round(total / len(valid), 1))
```

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