ANSWERS

A13.8 Project: the way out

Functional programming · A level · AQA 7517 4.12.1.1, Eduqas A500QS 1.4 · about 35 min

BugBotLab

What this lesson is about

Sweep the bay, decide with a pipeline of pure functions built by composition, map, filter and fold, then drive out.

Questions

5 marks in all

1. In the project, which of these are pure functions? [1 mark]

Tick every answer that is true.

- ☐ **A** offset(-15)
- ☐ **B** compose
- ☐ **C** room_of
- ☐ **D** sweep
- ☐ **E** the function passed to reduce to keep the pair with more room

Answer: A, B, C, E. sweep reads the sensor and turns the robot, so its result depends on the world and it changes the world.

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

```
from functools import reduce

def offset(a):
    return lambda x: x + a

def compose(g, f):
    return lambda x: g(f(x))

room_of = compose(offset(-15), lambda r: r[1])
rooms = tuple(map(lambda r: (r[0], room_of(r)), ((0, 30.0), (90, 52.0), (180, 52.0))))
best = reduce(lambda a, b: a if a[1] >= b[1] else b, rooms)
print(best)
```

Answer:

```
(90, 37.0)
```

The rooms are 15.0, 37.0 and 37.0. On the tie, >= keeps the earlier pair.

3. room_of takes a (heading, cm) pair and returns the room in cm. Which is the best description of its co-domain? [1 mark]

- ☐ **A** Real numbers
- ☐ **B** (heading, cm) pairs
- ☐ **C** Boolean
- ☐ **D** Headings from 0 to 315

> **Answer:** A. The co-domain is the set results come from; the pairs are its domain.

4. room_of = compose(offset(-15), cm_of). Which is applied to the pair first? [1 mark]

- ☐ **A** cm_of, then offset(-15)
- ☐ **B** offset(-15), then cm_of
- ☐ **C** Both at the same time
- ☐ **D** Whichever is quicker

> **Answer:** A. compose(g, f) is g ∘ f, which applies f first. Subtracting 15 from a pair would make no sense.

5. What is the main benefit of keeping the robot's side effects in a thin layer at the edges of the program? [1 mark]

- ☐ **A** The decision-making core is pure, so it can be tested with made-up readings, without a robot, and always gives the same answer
- ☐ **B** The robot drives faster
- ☐ **C** The program no longer needs to read the sensor
- ☐ **D** Side effects become impossible

> **Answer:** A. Sensing and driving still happen, but the logic between them can be reasoned about and tested on its own.

The task: the way out

Get BugBot out of the bay, following the brief. Every part of the program is defined here: - `MARGIN = 15` (cm) and `OPEN_CM = 25` (cm) are constants. - `sweep()` takes no arguments and returns a tuple of 8 `(heading, cm)` pairs, heading `0, 45, ..., 315` and cm the `distance()` reading taken facing that way, turning right 45 degrees at speed 30 after each reading. - `offset(amount)` returns a function of one number `x` that gives `x + amount`; `compose(g, f)` returns a function of one value `x` that gives `g(f(x))`. - `room_of` is `compose` applied to `offset(-MARGIN)` and a function that returns a pair's cm: given a `(heading, cm)` pair it returns the room in cm. - Map over the sweep to get `(heading, room)` pairs. Filter them to those with room greater than `OPEN_CM` and print their headings in sweep order, separated by spaces, as `open: <headings>`. - Fold the `(heading, room)` pairs to the one with the most room (the earlier one on a tie) and print `best: <heading> with <room> cm of room`, with the room rounded to one decimal place. - Turn right by the best heading at speed 30, and if the room is more than 0, drive forward by the room at speed 50. Use `map`, `filter` and `reduce`; do not use `max`, `min` or `sorted`. The robot must not touch anything.

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

MARGIN = 15
OPEN_CM = 25

def sweep():
    readings = ()
    return readings

readings = sweep()
print("open:")
```

The hint students can ask for: Build and test the pure part first with made-up readings, as in steps 2 and 3, before the robot moves at all. Then add the sweep, and check that the pairs it returns look like the table in step 1. The robot's last two commands only need the best pair.

A solution

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

MARGIN = 15
OPEN_CM = 25

def sweep():
    readings = ()
    for i in range(8):
        readings = readings + ((i * 45, distance()),)
        turn_right(30, angle=45)
    return readings

def offset(amount):
    return lambda x: x + amount

def compose(g, f):
```

```
    return lambda x: g(f(x))

room_of = compose(offset(-MARGIN), lambda pair: pair[1])

readings = sweep()
rooms = tuple(map(lambda r: (r[0], room_of(r)), readings))
open_ones = filter(lambda r: r[1] > OPEN_CM, rooms)
print("open:", " ".join(map(lambda r: str(r[0]), open_ones)))

best = reduce(lambda a, b: a if a[1] >= b[1] else b, rooms)
print("best:", best[0], "with", round(best[1], 1), "cm of room")

turn_right(30, angle=best[0])
if best[1] > 0:
    forward(50, distance=best[1])
```

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