ANSWERS

A13.4 Partial application and composition

Functional programming · A level · AQA 7517 4.12.1.4, Eduqas A500QS 1.4 · about 25 min

What this lesson is about

Why every Haskell function takes one argument, partial function application, functools.partial, and composing functions into a sensor pipeline.

- Every function takes one argument
- Partial application in Python
- Composition of functions
- A sensor pipeline

Questions

6 marks in all

1. In Haskell, mult x y = x * y and triple = mult 3. What is triple 7? [1 mark]

> **Answer:** 21. mult 3 is a partial application: a function that multiplies its argument by 3.

2. add :: Integer -> Integer -> Integer. What is the type of add 4? [1 mark]

- ☐ **A** Integer -> Integer
- ☐ **B** Integer
- ☐ **C** Integer -> Integer -> Integer
- ☐ **D** (Integer, Integer) -> Integer

> **Answer:** A. The arrow groups to the right, so add is Integer -> (Integer -> Integer): given one integer it returns a function from integer to integer.

3. f(x) = x + 2 and g(y) = y^3. What is (g ∘ f)(2)? [1 mark]

> **Answer:** 64. g ∘ f applies f first: f(2) = 4, then g(4) = 64.

4. f(x) = x + 2 and g(y) = y^3. What is (f ∘ g)(2)? [1 mark]

> **Answer:** 10. f ∘ g applies g first: g(2) = 8, then f(8) = 10. Composition is not commutative.

5. f: A → B and g: B → C. What is the type of g ∘ f? [1 mark]

- ☐ **A** A → C
- ☐ **B** C → A
- ☐ **C** B → B
- ☐ **D** A → B → C

> **Answer:** A. g ∘ f takes an argument from A, applies f to get a B, then g to get a C.

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

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

double = lambda x: x * 2
less5 = lambda x: x - 5
print(compose(double, less5)(10), compose(less5, double)(10))
```

> **Answer:**
>
> ```
> 10 15
> ```
>
> compose(double, less5) subtracts 5 first, giving 10; compose(less5, double) doubles first, giving 15.

The task: calibrate by composition

Build a calibration pipeline for the distance sensor from small curried functions: - `scale(factor)` returns a function of one number `cm` that gives `cm * factor`. - `offset(amount)` returns a function of one number `cm` that gives `cm + amount`. - `compose(g, f)` returns a function of one value `x` that gives `g(f(x))`. Using only those three, make `pipeline`: first `offset(-2)`, then `scale(10)`. Apply it to each reading in `RAW = (38.0, 51.5, 20.0)` and print the results on one line as `mm: 360.0 495.0 180.0`. Then compose the same two stages in the **wrong** order and print its results the same way as `wrong order: <three numbers>`. Round each result 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()

RAW = (38.0, 51.5, 20.0)

def pipeline(cm):
    return cm * 10 - 2

print("mm:", pipeline(RAW[0]))
```

The hint students can ask for: Each of scale and offset returns a function that is still waiting for its cm. compose returns a function too. Remember which argument of compose is applied first, then write the wrong order by swapping them.

A solution

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

RAW = (38.0, 51.5, 20.0)

def scale(factor):
    return lambda cm: cm * factor

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

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

pipeline = compose(scale(10), offset(-2))
wrong = compose(offset(-2), scale(10))

print("mm:", " ".join(map(lambda cm: str(round(pipeline(cm), 1)), RAW)))
print("wrong order:", " ".join(map(lambda cm: str(round(wrong(cm), 1)), RAW)))
```

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