ANSWERS

# A13.2 Function types and function application

Functional programming · A level · AQA 7517 4.12.1.1 · about 20 min

## What this lesson is about

Functions as mappings, function type f: A to B, domain and co-domain, and function application with arguments from a Cartesian product.

- A function is a mapping
- Function type, domain and co-domain
- Writing the type in code
- Function application
- Applying a type to the robot

## Questions

**6 marks in all**

**1.** A function has type f: A → B. What is B called? [1 mark]

- ☐ **A** The co-domain
- ☐ **B** The domain
- ☐ **C** The argument type
- ☐ **D** The Cartesian product

> **Answer:** A. A is the argument type, called the domain; B is the result type, called the co-domain.

**2.** is_even takes a whole number and returns True or False. What is its function type? [1 mark]

- ☐ **A** is_even: integer → Boolean
- ☐ **B** is_even: Boolean → integer
- ☐ **C** is_even: integer × Boolean → integer
- ☐ **D** is_even: integer → integer

> **Answer:** A. The argument type comes before the arrow and the result type after it.

**3.** Which statements about a function f: A → B are true? [1 mark]

Tick every answer that is true.

- ☐ **A** A and B are subsets of objects in some data type
- ☐ **B** Each value in A is mapped to exactly one value in B
- ☐ **C** Every value in B must be the result for some value in A
- ☐ **D** A is the set of results the function gives

> **Answer:** A, B. The co-domain is the set results are taken from; a function need not produce every value in it. The domain, not the co-domain, is the set of arguments.

**4.** add has type integer × integer → integer. What is integer × integer called? [1 mark]

> **Answer:** Cartesian product. integer × integer is the Cartesian product of the integers with itself: the set of all pairs of integers.

**5.** In function application, how many arguments does add(3, 4) strictly take, if add: integer × integer → integer? [1 mark]

- ☐ **A** One, the pair (3, 4)
- ☐ **B** Two, 3 and 4
- ☐ **C** None, it is a constant
- ☐ **D** Three, 3, 4 and the result

> **Answer:** A. The domain is the Cartesian product integer × integer, so its single argument is a pair.

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

```
def compass(degrees):
    return "NESW"[((degrees + 45) % 360) // 90]

print(compass(44), compass(45), compass(224), compass(315))
```

> **Answer:**
>
> ```
> N E S N
> ```
>
> Adding 45 and dividing by 90 gives the quarter: 44 is still N, 45 becomes E, 224 is S, and 315 wraps round to N.

## The task: the compass type

Write `compass` with its type hints, exactly as `def compass(degrees: int) -> str:`. Its domain is the whole numbers 0 to 359, and it returns `"N"` for 0 to 44 and 315 to 359, `"E"` for 45 to 134, `"S"` for 135 to 224 and `"W"` for 225 to 314. Then: 1. Apply `compass` to **every** value in its domain, collect the different results, and print them sorted alphabetically and separated by spaces, as `range: <letters>`. Work the letters out; do not type them into the `print`. 2. Turn the robot right by 90 degrees four times. After each turn, print `facing <letter>`, where the letter is `compass` applied to the robot's `heading()` rounded to a whole number and taken MOD 360.

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

def compass(degrees):
    return "N"

print("range:", compass(0))
```

**The hint students can ask for:** Work out a way to turn any whole number of degrees into a quarter number 0 to 3, so the four ranges line up with N, E, S and W; check the edges 44, 45, 314 and 315. For the range, a set keeps only the different results of applying the function to every value in the domain.

## A solution

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

def compass(degrees: int) -> str:
    return "NESW"[((degrees + 45) % 360) // 90]

results = set(map(compass, range(360)))
print("range:", " ".join(sorted(results)))

for i in range(4):
    turn_right(30, angle=90)
    print("facing", compass(round(heading()) % 360))
```

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