WORKSHEET

A13.3 First-class objects and higher-order functions

Functional programming · A level · AQA 7517 4.12.1.2, Eduqas A500QS 1.4 · about 20 min

Name _____ Class _____ Date _____

What this lesson is about

Functions as values in variables, lists and dictionaries, passed as arguments and returned as results, lambdas, closures and a robot command table.

- First-class objects
- Functions as arguments
- Functions as results
- Higher-order functions
- A command table

Questions

6 marks in all

1. Which of these can a first-class object do? [1 mark]

Tick every answer that is true.

- ☐ **A** Appear in an expression
- ☐ **B** Be assigned to a variable
- ☐ **C** Be passed as an argument
- ☐ **D** Be returned from a function call
- ☐ **E** Only be used once

2. What makes a function higher-order? [1 mark]

- ☐ **A** It takes a function as an argument, returns a function as its result, or both
- ☐ **B** It is defined before the other functions in a program
- ☐ **C** It calls itself
- ☐ **D** It has more than two parameters

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

```
def apply_twice(f, x):
    return f(f(x))

print(apply_twice(lambda n: n * 3, 2))
```

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

```
def make_adder(n):
    return lambda x: x + n

fs = [make_adder(5), make_adder(-1), abs]
print([f(-3) for f in fs])
```

5. double is a function. What does the line f = double do? [1 mark]

- ☐ **A** Makes f refer to the function double, so f(4) can be called
- ☐ **B** Calls double and stores its result in f
- ☐ **C** Causes an error, because double needs an argument
- ☐ **D** Makes a copy of double's code with a new name that cannot be called

6. A Haskell function has type applyTwice :: (a -> a) -> a -> a. What is its first argument? [1 mark]

- ☐ **A** A function from a value of type a to a value of type a
- ☐ **B** A value of type a
- ☐ **C** A list of type a
- ☐ **D** A pair of values of type a

The task: the command table

Drive a route from a table of functions, with **no** `if` or `elif` anywhere in the program: the table does the choosing. - `drive_forward(cm)` drives forward `cm` centimetres at speed 50, and `drive_back(cm)` drives backward `cm` centimetres at speed 50. - `make_turn(direction)` takes `"left"` or `"right"` and **returns a new function** of one parameter, `degrees`, that turns that way by `degrees` at speed 30. - `ACTIONS` is a dictionary from the letters `"F"`, `"B"`, `"L"` and `"R"` to `drive_forward`, `drive_back`, `make_turn("left")` and `make_turn("right")`. - `run_route(actions, route)` takes a table like `ACTIONS` and a list of commands, each a letter followed by a whole number such as `"F25"`. For each command in order it prints the letter and the number separated by a space, such as `F 25`, and applies the letter's function to the number. Call `run_route(ACTIONS, ["F25", "R90", "F20", "L90", "B10"])`.

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

def run_route(route):
    for command in route:
        letter = command[0]
        amount = int(command[1:])
        if letter == "F":
            forward(50, distance=amount)
        elif letter == "R":
            turn_right(30, angle=amount)

run_route(["F25", "R90", "F20", "L90", "B10"])
```

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

Do it on the robot

www.bugbotlab.com/learn/a13-3-first-class-and-higher-order-functions/

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

Challenges

1. Add a command `"S"` that sidesteps right, using `right()`, by changing only the table.
2. Write `make_repeat(action, times)` that returns a function doing `action` `times` times with the same amount. Use it to add `"D"` for "forward, twice".
3. Write the type of `make_turn` in Haskell-style notation. Which part of it is a function type?