



A13.1 The functional paradigm

Functional programming · A level · OCR H446 1.2.4, AQA 7517 4.12.2.1,
Eduqas A500QS 1.4 · about 20 min

Name _____

Class _____

Date _____

What this lesson is about

Side effects, pure functions and referential transparency, immutability and statelessness, with the robot's side effects kept at the edges.

- A function, in the mathematical sense
- Immutability
- Statelessness
- Where the robot fits

Questions

6 marks in all

1. Which best describes a pure function? [1 mark]

- ☐ A Its result depends only on its arguments and it has no side effects
- ☐ B It is written without any if statements
- ☐ C It never takes more than one argument
- ☐ D It is defined inside a class

2. Which of these are side effects of a function? [1 mark]

Tick every answer that is true.

- ☐ A Printing a message
- ☐ B Changing a global variable
- ☐ C Returning a value
- ☐ D Reading the distance sensor
- ☐ E Working out the square root of its argument

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

```
calls = 0

def next_id(name):
    global calls
    calls = calls + 1
    return name + str(calls)

print(next_id("bot"), next_id("bot"))
```

4. What does this program print?

[1 mark]

```
a = [1, 2]
b = a
a = a + [3]
print(b)
```

5. A call to a pure function can always be replaced by the value it returns without changing what the program does. What is this property called?

[1 mark]

6. What does statelessness mean in functional programming?

[1 mark]

- ☐ A There is no program state that changes as the program runs: a name, once given a value, keeps it
- ☐ B The program has no variables at all
- ☐ C The program cannot store data in files
- ☐ D Functions cannot take parameters

The task: make it pure

The starter has two impure functions. Rewrite both as pure functions, with no `global` and nothing changed in place: - `count_clear(readings, limit)`: `readings` is a tuple of distances in cm (floats), `limit` a number of cm. It **returns** how many readings are greater than `limit`, and gives the same answer however many times it is called. - `add_reading(log, cm)`: `log` is a tuple of floats and `cm` a float. It **returns** a new tuple with `cm` added at the end, and leaves `log` exactly as it was. The main program (keep it as it is) prints `clear: 2` twice, then `before: (31.0, 51.0, 15.0, 44.0)` and `after: (31.0, 51.0, 15.0, 44.0, 60.0)`. Write the functions in the stateless style: no `global`, no `for` or `while` loop (use recursion, as `total_of` does), and no `append`, `extend` or `+=` anywhere.

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

READINGS = (31.0, 51.0, 15.0, 44.0)

count = 0
def count_clear(readings, limit):
    global count
    for cm in readings:
        if cm > limit:
            count = count + 1
    return count

def add_reading(log, cm):
    log = list(log)
    log.append(cm)
    return tuple(log)

print("clear:", count_clear(READINGS, 40))
print("clear:", count_clear(READINGS, 40))
new_log = add_reading(READINGS, 60.0)
print("before:", READINGS)
print("after:", new_log)
```

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



Do it on the robot

www.bugbotlab.com/learn/a13-1-the-functional-paradigm/

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

Challenges

1. Write a pure function `furthest(readings)` that returns the largest reading, using recursion and no loop.
2. Which of these are pure: `len`, `print`, `random.randint`, `abs`, `input`, `heading`? Say why for each.
3. Write `add_reading` in Haskell, using `++` to join lists. Why does Haskell not need a separate "copy" step?