ANSWERS

5.5 Brains

Behaviours · Robot club · about 20 min

What this lesson is about

From a loop to think(me) and me.memory, with a practice arena.

- The loop is outside
- Escape, as a loop
- The same thing as a brain

Questions

6 marks in all

1. How often does the arena call your `think(me)` function? [1 mark]

- ☐ **A** Ten times a second
- ☐ **B** Once, at the start of the game
- ☐ **C** Only when another robot comes close
- ☐ **D** As fast as the computer can

> **Answer:** A. The arena owns the loop and the waiting. `think` is the body of the loop, run ten times a second.

2. Why does a brain keep its state in `me.memory` rather than in a normal variable? [1 mark]

- ☐ **A** Local variables are forgotten when the function returns
- ☐ **B** `me.memory` is faster
- ☐ **C** Normal variables cannot hold words
- ☐ **D** The arena reads `me.memory` to score you

> **Answer:** A. Each call starts fresh. Anything that has to survive to the next tick goes in `me.memory` .

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

```
class Me:
    def __init__(self):
        self.memory = {}

def think(me):
    count = me.memory.get("count", 0)
    me.memory["count"] = count + 1

def forgetful(me):
    count = 0
    count += 1
    return count

me = Me()
for i in range(3):
    think(me)
    n = forgetful(me)
print(me.memory["count"])
print(n)
```

Answer:

```
3
1
```

`think` saves its count in memory, so it reaches 3. `forgetful` starts from 0 every call, so it is always 1.

4. On the very first tick, `me.memory` is empty. What does `me.memory.get("state", "go")` give? [1 mark]

- ☐ **A** `"go"`, the default
- ☐ **B** `None`
- ☐ **C** An error, because the key is missing
- ☐ **D** An empty string

Answer: A. `get` returns the second argument when the key is not there yet, which is how the machine picks its starting state.

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

```
samples = [(20, 0), (25, 45), (60, 90), (80, 135), (70, 180), (30, 225), (20, 270)]
opens = [h for d, h in samples if d > 40]
target = opens[len(opens) // 2]
print(opens, target)
```

Answer:

```
[90, 135, 180] 135
```

Headings with a reading over 40 cm are the gap. The middle one of those is 135, the centre of the opening.

6. The escape program aims at the middle of the open headings, not the single longest reading. Why? [1 mark]

- ☐ **A** The longest reading points at a corner of the opening
- ☐ **B** The longest reading is always wrong
- ☐ **C** The middle is quicker to work out
- ☐ **D** The distance sensor cannot read long distances

> **Answer:** A. Through a gap, the furthest thing seen is usually past one edge. Aiming at the middle keeps you clear of both sides.

The task: escape the box

Three walls around you. Find the gap with the distance sensor, drive out and into the green zone, touching nothing.

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

# as long as the thing ahead is further than 15 cm
while distance() > 15:
    # drive forward at 50 (keeps going until the next command)
    forward(50)
    # pause 0.1 s (the robot keeps doing what it was told)
    wait(0.1)
# all motors off
stop()
print("wall at", distance())
```

The hint students can ask for: Three walls around you; the way out is on one side. Find it with the distance sensor, drive out, and into the green zone, without touching a wall.

A solution

```
from bugbot import *
connect()
def wrapped(h):
    return (h + 180) % 360 - 180
state = "look"
samples = []
for tick in range(290):
    if state == "look":
        samples.append((distance(), heading()))   # a full turn of readings
        turn_right(50)
        if len(samples) >= 65:
            opens = [h for d, h in samples if d > 40]   # headings with room: the gap
            target = opens[len(opens) // 2]              # the middle of the gap, not the
longest ray
            state = "turn"
    elif state == "turn":
        error = wrapped(target - heading())
        if abs(error) < 4:
            state = "go"
        else:
            drive(0, 0, max(-40, min(40, error * 3)))
    elif state == "go":
        x, y = position()
        if y < -32:
            break
        drive(60, 0, wrapped(target - heading()) * 3)
    wait(0.1)
stop()
```

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