ANSWERS

A8.4 De Morgan's laws

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.5.1, Eduqas A500QS 1.2 · about 20 min

What this lesson is about

Breaking the bar and changing the sign, NAND and NOR, and rewriting the robot's loop condition.

- The two laws
- Break the bar, change the sign
- More than two inputs, and nested brackets
- Why NAND and NOR are universal
- De Morgan in your programs

Questions

6 marks in all

1. By De Morgan's law, ¬(A ∧ B) equals which expression? [1 mark]

- ☐ **A** ¬A ∨ ¬B
- ☐ **B** ¬A ∧ ¬B
- ☐ **C** A ∨ B
- ☐ **D** ¬A ∧ B

> **Answer:** A. Invert each term and change AND to OR.

2. By De Morgan's law, ¬(A ∨ B ∨ C) equals which expression? [1 mark]

- ☐ **A** ¬A ∧ ¬B ∧ ¬C
- ☐ **B** ¬A ∨ ¬B ∨ ¬C
- ☐ **C** A ∧ B ∧ C
- ☐ **D** ¬A ∨ B ∨ C

> **Answer:** A. The law extends to any number of terms: invert each and change every OR to AND.

3. What does ¬(¬A ∨ ¬B) simplify to? [1 mark]

- ☐ **A** A ∧ B
- ☐ **B** A ∨ B
- ☐ **C** ¬A ∧ ¬B
- ☐ **D** ¬(A ∧ B)

> **Answer:** A. De Morgan gives ¬¬A ∧ ¬¬B, and double negation gives A ∧ B.

4. Which Python condition means the same as not (close or hit)? [1 mark]

- ☐ **A** not close and not hit
- ☐ **B** not close or not hit
- ☐ **C** not close and hit
- ☐ **D** close and not hit

> **Answer:** A. NOT (X OR Y) is (NOT X) AND (NOT Y).

5. In AQA notation, how do $\overline{A} \cdot \overline{B}$ (a bar over each letter) and a single bar over the whole of $A \cdot B$ differ? [1 mark]

- ☐ **A** The first is NOR and the second is NAND
- ☐ **B** They are the same function
- ☐ **C** The first is NAND and the second is NOR
- ☐ **D** The first is XOR and the second is AND

> **Answer:** A. $\overline{A} \cdot \overline{B}$ is $\neg A \wedge \neg B$, which De Morgan shows is $\neg(A \vee B)$, NOR. A long bar over $A \cdot B$ is NAND.

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

```
result = []
for a in [0, 1]:
    for b in [0, 1]:
        result.append((1 - (a & b)) == ((1 - a) | (1 - b)))
print(all(result))
```

> **Answer:**
>
> ```
> True
> ```
>
> De Morgan's law holds on all four rows, so every comparison is True.

The task: De Morgan at the wall

The robot starts facing a wall. First print four check lines, one for each combination of `close` and `hit` (each 0 or 1, `close` in the outer loop), in exactly this form: `close=0 hit=1 original=0 rewritten=0` where `original` is NOT (close OR hit) and `rewritten` is your De Morgan version, (NOT close) AND (NOT hit), each worked out as 0 or 1 by a function of your own. Then drive the robot towards the wall and stop in the band before it. The loop must be a `while` loop whose condition reads `distance()`, stops when the distance is under 20 cm or `bumped()` is true, and is written using De Morgan so that it joins its two parts with `and not`, with no `not` in front of a bracket.

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

while not (distance() < 20 or bumped()):
    forward(50)
    wait(0.1)
stop()
```

The hint students can ask for: De Morgan turns NOT (X OR Y) into (NOT X) AND (NOT Y). Apply it to the loop condition, and remember that NOT (distance < 20) can also be written as a comparison the other way round. Print the four check lines before you drive.

A solution

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

def original(close, hit):
    return 1 - (close | hit)

def rewritten(close, hit):
    return (1 - close) & (1 - hit)

for close in [0, 1]:
    for hit in [0, 1]:
        print(f"close={close} hit={hit} original={original(close, hit)} rewritten=
{rewritten(close, hit)}")

while distance() >= 20 and not bumped():
    forward(50)
    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.