Module 4: Logical Operations

Prof. Dr. Florian Haselbeck
Memoona Shakeel

Weihenstephan-Triesdorf University of Applied Sciences

Learning outcomes

Understand logical reasoning used in algorithms and programming decisions, and write your own logical expressions.

Key Topics

- True / False recap
- AND, OR, NOT, XOR operators
- Comparison operators
- if / elif / else
- Complex expressions
- Operator precedence

Boolean Logic

Key idea: A logical expression is any statement that evaluates to exactly True or False. This is the basis of every decision a computer makes.

Where booleans come from?

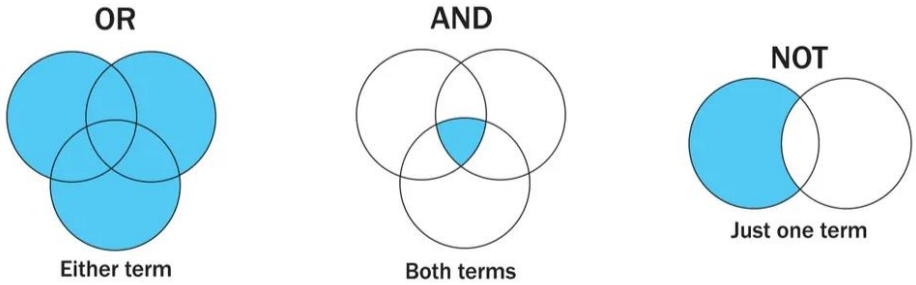
```
Comparison: temp < 2.0 → True
Equality: crop == "wheat" → True/False
Function return: sensor_ok() → True/False
```

Combining Conditions:

```
AND: both must be True
OR:  at least one must be True
NOT: flip True → False, False → True
```

Python uses words:

```
Python: and  or  not  (lowercase)
C/Java: &&  ||  !
Maths/logic: ∧  ∨  ¬
```



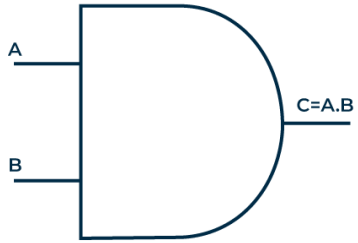

```python
# AND → both conditions must be True
print(True and True)     # True
print(True and False)    # False

# OR → at least one must be True
print(True or False)     # True
print(False or False)    # False

# NOT → reverses the value
print(not True)          # False
print(not False)         # True
```

AND (both conditions must be True)

Rule: **A** AND **B** is True only when BOTH A is True AND B is True. If either one is False, the result is False.

Short-circuit: Python stops at the first False, if the first part is False, it does not check the second

```python
#  Farm conditions (digital check system)
is_sunny = True
has_water = True

#  Decision: can we start irrigation system?
start_irrigation = is_sunny and has_water

print("Start irrigation:", start_irrigation) # True

# Farm system check
is_sunny = False
has_water = True

# Python stops at "is_sunny" because it's False
start_irrigation = is_sunny and has_water

print(start_irrigation)   # False
```



Truth Table of AND gate

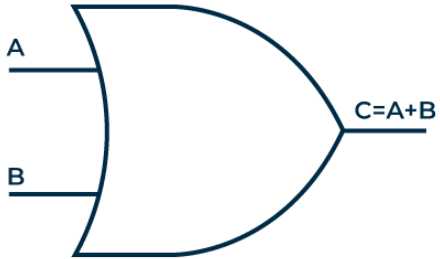
A	B	C=A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR (at least one condition must be True)

Rule: A OR B is True when EITHER A is True OR B is True (or both). Only False when BOTH are False.

Short-circuit: Python stops at the first True, if the first part is True, it does not check the second

```python
# Farm system
has_backup_power = True
is_main_power_on = False

# Python stops at first True (has_backup_power)
system_running = has_backup_power or is_main_power_on

print(system_running)    # True
```



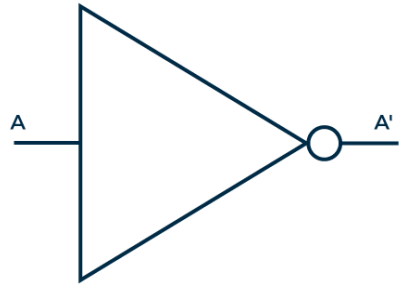
Truth Table of OR gate

A	B	C=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT (flips the boolean value)

Rule: NOT A is True when A is False, and False when A is True. It inverts > negates > the condition.

```python
# Weather condition
is_raining = False

# NOT flips it
can_work = not is_raining

print(can_work)   # True (can work because it's NOT raining)

# Farm status
farm_open = True

# NOT flips the value
print(not farm_open)   # False (farm is NOT open)
```



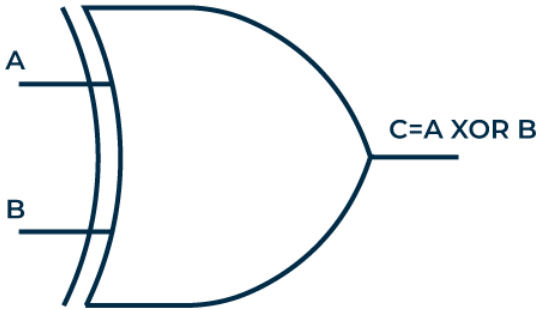
Truth Table of NOT gate

INPUT	OUTPUT
1	0
0	1

XOR (exactly one must be True Exclusive OR)

Rule: A XOR B is True when exactly ONE of A or B is True, but NOT both, and NOT neither.

```python
# Farm backup system
solar_on = True
generator_on = False

# Only ONE system should be ON
power = solar_on ^ generator_on

print("Power working:", power)
```



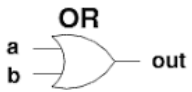
Truth Table of XOR gate

A	B	C=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

All operators at a glance

```python
# Farm conditions
sunny = True
water_available = False

# AND → both must be True
print("AND:", sunny and water_available)
# False → because water is False

# OR → at least one True
print("OR:", sunny or water_available)
# True → because sunny is True

# NOT → flips value
print("NOT sunny:", not sunny)
# False → because sunny was True

# XOR → exactly ONE must be True
print("XOR:", sunny ^ water_available)
# True → one is True, one is False
```

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



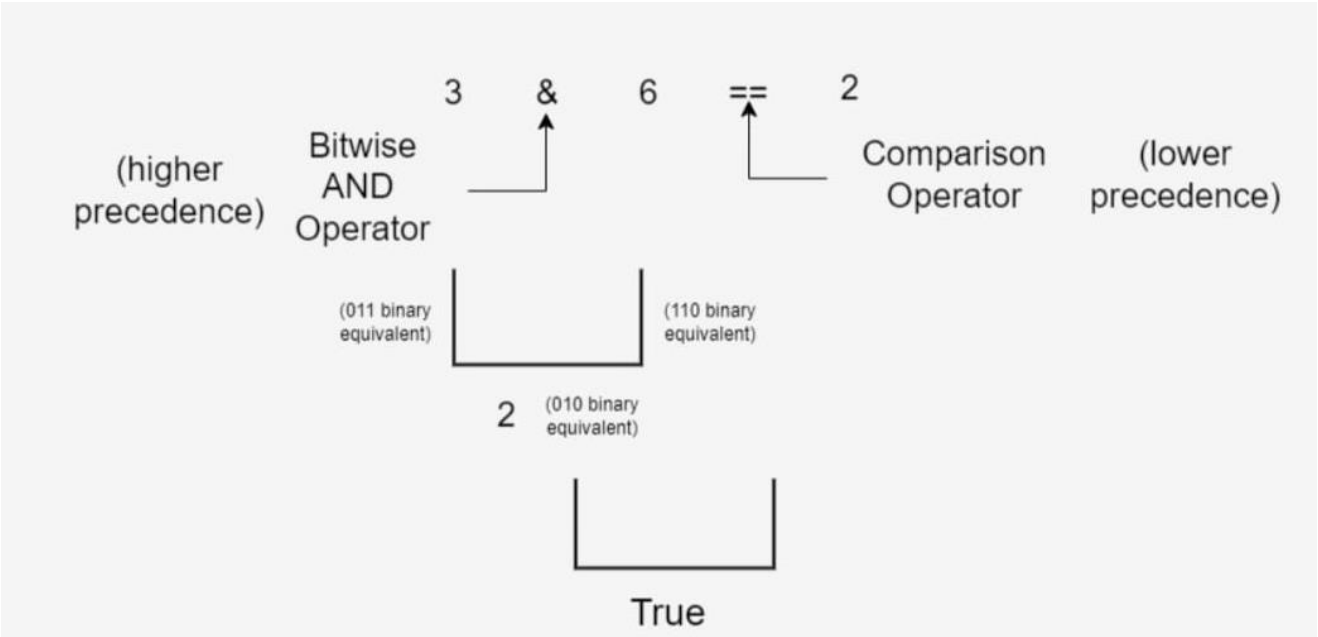
a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

in	out
0	1
1	0

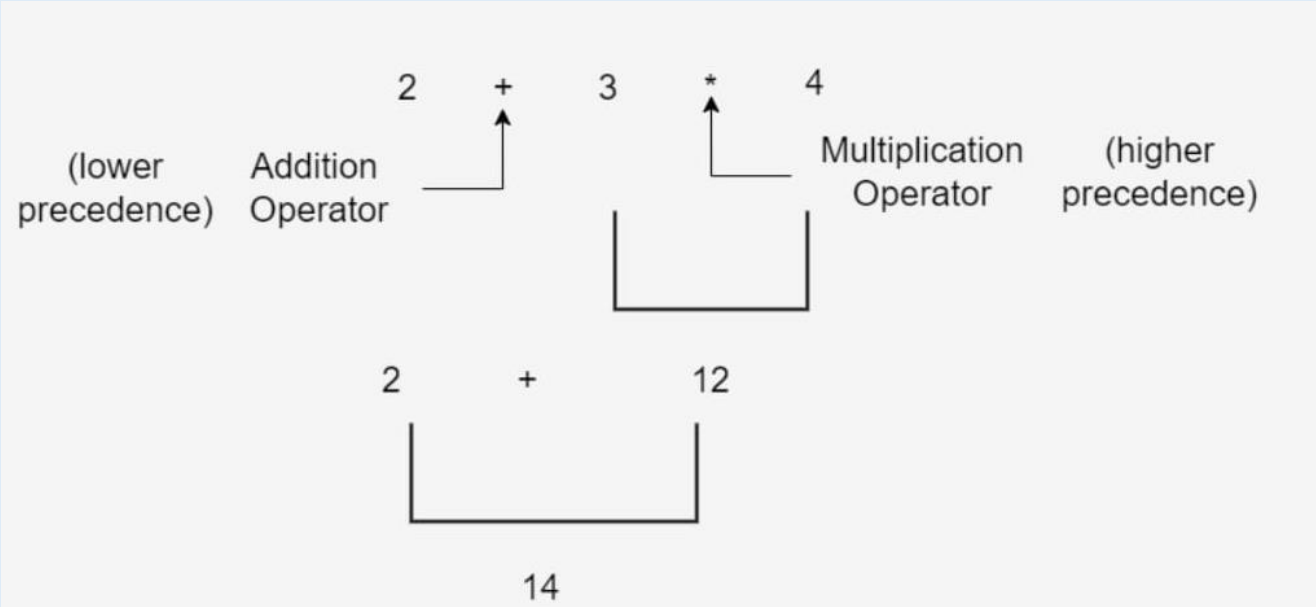
Operator precedence

Operator precedence (highest → lowest): Comparisons → NOT → AND → OR



Operator precedence

Easy memory trick
First: Math (*/+−)
Then: Comparisons (>, <)
Then: NOT
Then: AND
Then: OR

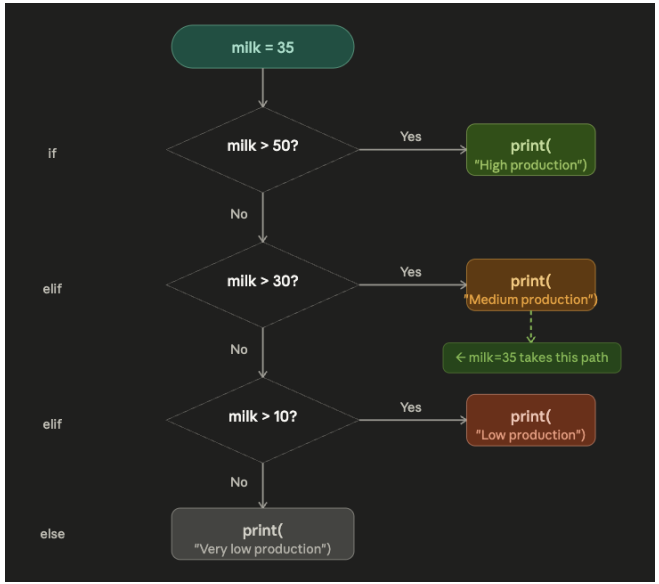


Conditional statements: if / elif / else

Key idea: A conditional statement runs a block of code only when a logical expression is True. This is how a program makes decisions.

- Only one block runs
- Python checks from top to bottom
- Stops when the first condition is True

```python
#  Farm milk production (liters per day)
milk = 35

# Decision making using if / elif / else
if milk > 50:
    print("High production ")
elif milk > 30:
    print("Medium production ")
elif milk > 10:
    print("Low production ")
else:
    print("Very low production ")
```



Complex logical expressions

Key idea: Real programs combine multiple conditions using AND, OR, NOT together. Evaluate step by step, innermost parentheses first.

```python
# Farm system conditions
sunny = True
water_available = False
fuel_available = True

# Combined logical expression
result = (sunny and water_available) or (not fuel_available)

print("Farm system result:", result)
```

```python
# Smart Farm System Conditions

temp = 1.5
frost_sensor = False

soil_dry = True
is_raining = False
tank_empty = False

ndvi = 0.25
field = {"irrigated": True}

temp_a = 85.0


#  1. Weather / frost alert
if temp < 2.0 or frost_sensor:
    print("ALERT: Cold weather or frost detected!")

#  2. Irrigation system control
if soil_dry and not is_raining and not tank_empty:
    print("ACTION: Opening water valve...")

#  3. Crop health warning
if ndvi < 0.3 and field["irrigated"]:
    print("WARNING: Crop health issue detected!")

#  4. Temperature safety check
if temp_a > 80.0 or temp_a < -40.0:
    print("MAINTENANCE: Check system temperature!")
```

Writing your own logical expressions text to code

"Send alert if the temperature drops below 2°C OR the frost sensor is triggered"

"Irrigate only if the soil is dry AND it is not raining AND the tank is not empty"

"Log a warning if NDVI is below 0.3 AND the field has been recently irrigated"

"Trigger maintenance if either temperature sensor gives an impossible reading"

Common mistakes and how to avoid them

These are the errors one makes most often with logical operations.

✗
```
if temp < 2.0 and 10.0:
```
→
```
if temp < 2.0 and temp < 10.0:
```
Python reads: True and 10.0, 10.0 is always truthy! Must repeat the variable.

✗
```
if a = True:
```
→
```
if a == True:  or just  if a:
```
Single = is assignment, not comparison. Python raises SyntaxError.

✗
```
if not a and b == False:
```
→
```
if not a and not b:
```
Mixing not with == False is redundant and harder to read.

✗
```
if temp < 2 or 5:
```
→
```
if temp < 2 or temp < 5:
```
5 is always truthy, this condition is always True! Repeat the variable.

Overview of key content

- AND requires both True · OR requires one True · NOT flips · XOR requires exactly one True
- Comparison operators always return a boolean: == != < > <= >=
- if / elif / else: exactly one block runs based on the first True condition
- Precedence order: NOT first, then AND, then OR, use parentheses to be explicit

Thank you!

Contact Detail

Prof. Dr. Florian Haselbeck

florian.haselbeck@hswt.de

Raum D1.230a

For any questions, please feel free to use the Moodle-Course Forum!