

# **Fundamentos de Programação**

António J. R. Neves  
João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática  
Universidade de Aveiro

# Summary

- Boolean expressions
  - The bool type
  - Relational operators
  - Logical operators
  - Properties
- Conditional execution
  - If statement
  - If-else
  - If-elif-else
- Conditional expression

# Boolean expressions

- A **boolean expression** is an expression that is either true or false.

```
>>> n = 5          # this IS NOT a boolean expression!
>>> n == 5         # this IS a boolean expression!
True
>>> 6 == n         # this is another boolean expression.
False
```

- True and False are special values that belong to the type bool.
- Boolean values may be stored in variables.

```
>>> isEven = n%2==0
```

- May be converted to string.

```
>>> str(isEven)
'False'
```

- Or to integer.

```
>>> int(False)    # 0
>>> int(True)     # 1
```

## Null and empty values convert to False:

```
>>> bool(0)        # False
>>> bool(0.0)      # False
>>> bool('')       # False
>>> bool([])       # False
```

## Other values convert to True:

```
>>> bool(1)        # True
>>> bool('False')  # True (surprise!)
>>> bool([False])  # True (surprise?)
```

# Relational and logical operators

- **Relational operators** produce boolean results:

```
x == y      # x is equal to y
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
x < y < z   # x is less than y and y is less than z (cool!)
```

- There are three **logical operators**: and, or, not.

```
x>=0 and x<10      # x is between 0 and 10 (exclusive)
0<=x and x<10      # same thing
x==0 or not isEven and y/x>1
```

# Properties

- Remember these properties:

$x == y$	$\Leftrightarrow$	$\text{not } (x != y)$	$\Leftrightarrow$	$y == x$
$x != y$	$\Leftrightarrow$	$\text{not } (x == y)$	$\Leftrightarrow$	$y != x$
$x > y$	$\Leftrightarrow$	$\text{not } (x \leq y)$	$\Leftrightarrow$	$y < x$
$x \leq y$	$\Leftrightarrow$	$\text{not } (x > y)$	$\Leftrightarrow$	$y \geq x$

- And these (where A, B, C are boolean):

$\text{not } (\text{not } A)$	$\Leftrightarrow$	$A$
$\text{not } (A \text{ and } B)$	$\Leftrightarrow$	$(\text{not } A) \text{ or } (\text{not } B)$
$\text{not } (A \text{ or } B)$	$\Leftrightarrow$	$(\text{not } A) \text{ and } (\text{not } B)$
$A \text{ or } B$	$\Leftrightarrow$	$B \text{ or } A$
$A \text{ and } B$	$\Leftrightarrow$	$B \text{ and } A$
$A \text{ or } (B \text{ and } C)$	$\Leftrightarrow$	$(A \text{ or } B) \text{ and } (A \text{ or } C)$
$A \text{ and } (B \text{ or } C)$	$\Leftrightarrow$	$(A \text{ and } B) \text{ or } (A \text{ and } C)$

# Precedence rules

- Arithmetic > relational > not > and > or.

$x \leq 1 + 2 * y ** 3 \text{ or } n \neq 0 \text{ and not } 1/n \leq y$

$(\underline{x \leq 1 + 2 * y ** 3}) \text{ or } (\underline{n \neq 0 \text{ and not } 1/n \leq y})$

$(x \leq (\underline{1 + 2 * y ** 3})) \text{ or } ((\underline{n \neq 0}) \text{ and } (\underline{\text{not } 1/n \leq y}))$

$(x \leq (1 + (\underline{2 * y ** 3}))) \text{ or } ((n \neq 0) \text{ and } (\text{not } (\underline{1/n \leq y})))$

$(x \leq (1 + (2 * (\underline{y ** 3})))) \text{ or } ((n \neq 0) \text{ and } (\text{not } ((\underline{1/n}) \leq y)))$

# Short-circuit evaluation

- Operators `and` and `or` only evaluate the second operand if needed!

`A and B`     # if A is false then A, otherwise B

`A or B`     # if A is true then A, otherwise B

- This is called **short-circuit evaluation**.
- It can be very useful:

`1/n>2 and n!=0`     # `ZeroDivisionError` if `n==0`

`n!=0 and 1/n>2`     # False if `n==0`, `1/n` not evaluated

`n==0 or 3/n<4`     # True if `n==0`, `3/n` not evaluated

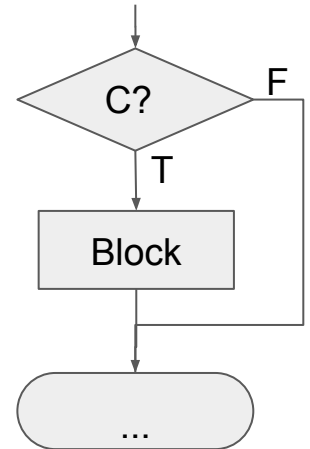
- But notice that the order of the operands is important!

# Conditional execution 1: simple **if**

- **Conditional statements** allow the program to check conditions and change its behavior accordingly.

- The simplest form is the `if` statement:

```
if condition:  
    block_of_statements  
...
```



- The *condition* should be a boolean expression. (It may be of any type, and is implicitly converted to `bool`, but this is confusing and should be avoided.)
- The block must have one or more *indented* statements.
- The *condition* is evaluated. If true, the *block of statements* is executed. If not, execution continues after the block.



# Example 1

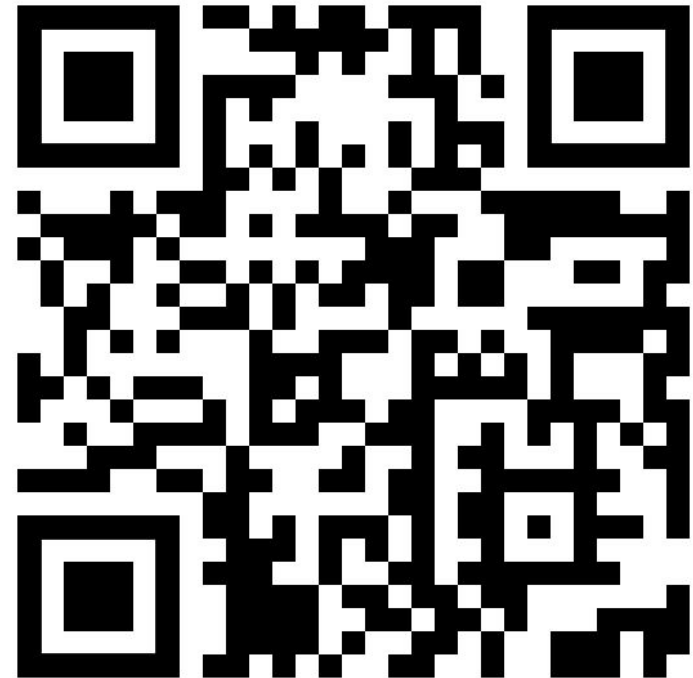
- What is the output if  $N = 3$ ?
- What if  $N = 4$ ,  $N = 13$  or  $N = 14$ ?

```
N = int(input("N? "))  
  
if N > 10:  
    print("A")  
  
if N % 2 == 0:  
    print("B")  
  
print("END")
```

[Play ▶](#)

Answer questions on:

<https://forms.gle/cfjsNAHt8xov5VGP7>



[Edit](#) [Responses](#)

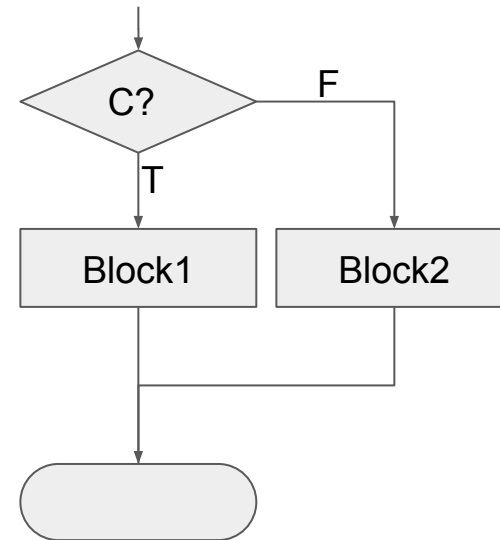
# Conditional execution 2: **if - else**

- A second form of the `if` statement is alternative execution, in which there are two alternative paths and the condition determines which one gets executed.

```
x = 3
```

```
if x%2 == 0:  
    R = 'even'  
else:  
    R = 'odd'
```

```
print(x, 'is', R)
```

[Play ►](#)

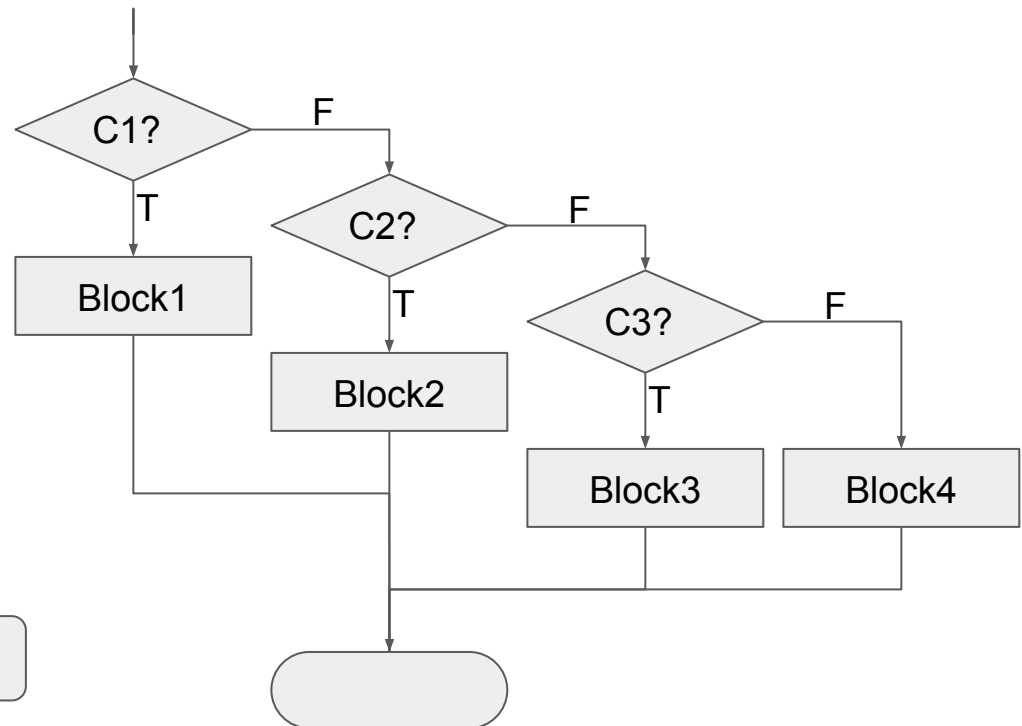
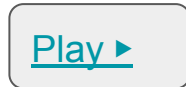
# Conditional execution 3: **if - elif - else**

- Sometimes there are more than two alternatives and we need more than two branches (chained conditional).

```
x = 12
```

```
if x < 10:  
    mark = 'Poor'  
elif x < 13:  
    mark = 'Fair'  
elif x < 17:  
    mark = 'Good'  
else:  
    mark = 'Excelent'
```

```
print(mark)
```



# Conditional statement semantics

- Which conditions select each block of statements?

```
if C1:
    Block1      ← Block1 is executed iff C1
elif C2:
    Block2      ← Block2 is executed iff  $\neg C1 \wedge C2$ 
elif C3:
    Block3      ← Block3 is executed iff  $\neg C1 \wedge \neg C2 \wedge C3$ 
else:
    Block4      ← Block4 is executed iff  $\neg C1 \wedge \neg C2 \wedge \neg C3$ 

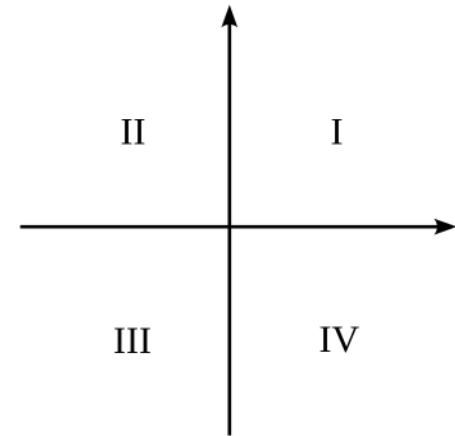
Rest           ← is always executed
```

# Nested conditional statements

- Conditional statements may be nested within each other.

```
if y > 0:
    if x > 0:
        quadrant = 1
    else:
        quadrant = 2
else:
    if x < 0:
        quadrant = 3
    else:
        quadrant = 4
```

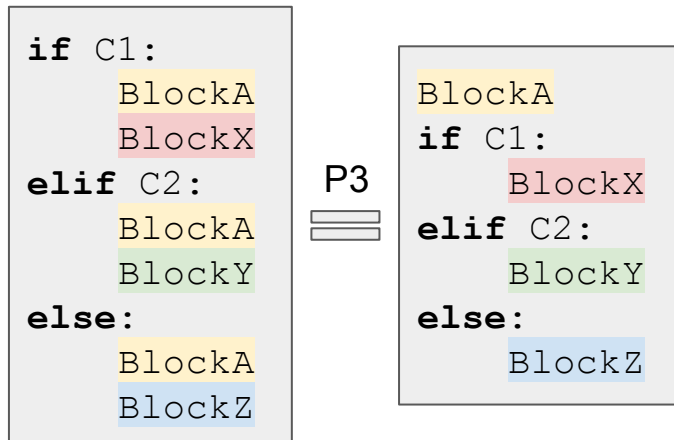
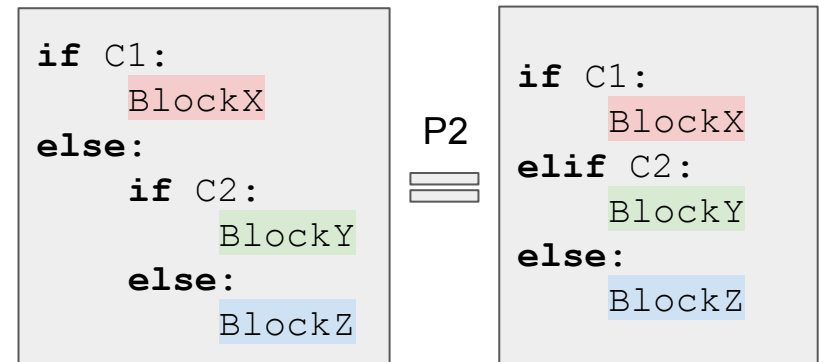
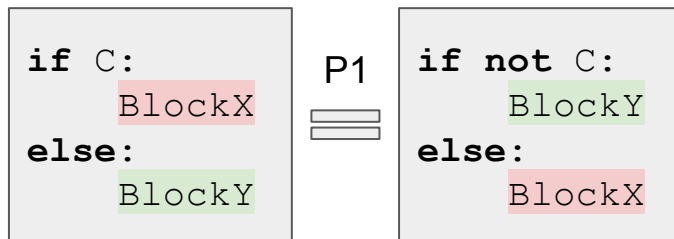
[Play ▶](#)



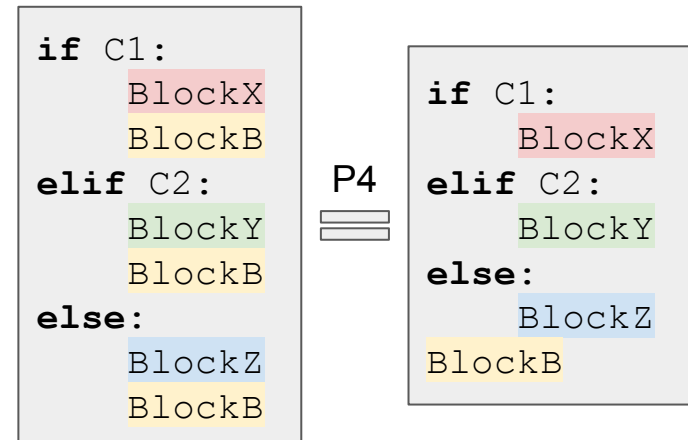
- Although the indentation makes the structure apparent, deeply nested conditionals become difficult to read.
- If possible, apply equivalence properties to simplify nested conditional statements.

# Program equivalence properties

For *well-behaved* blocks of statements, the following properties apply.



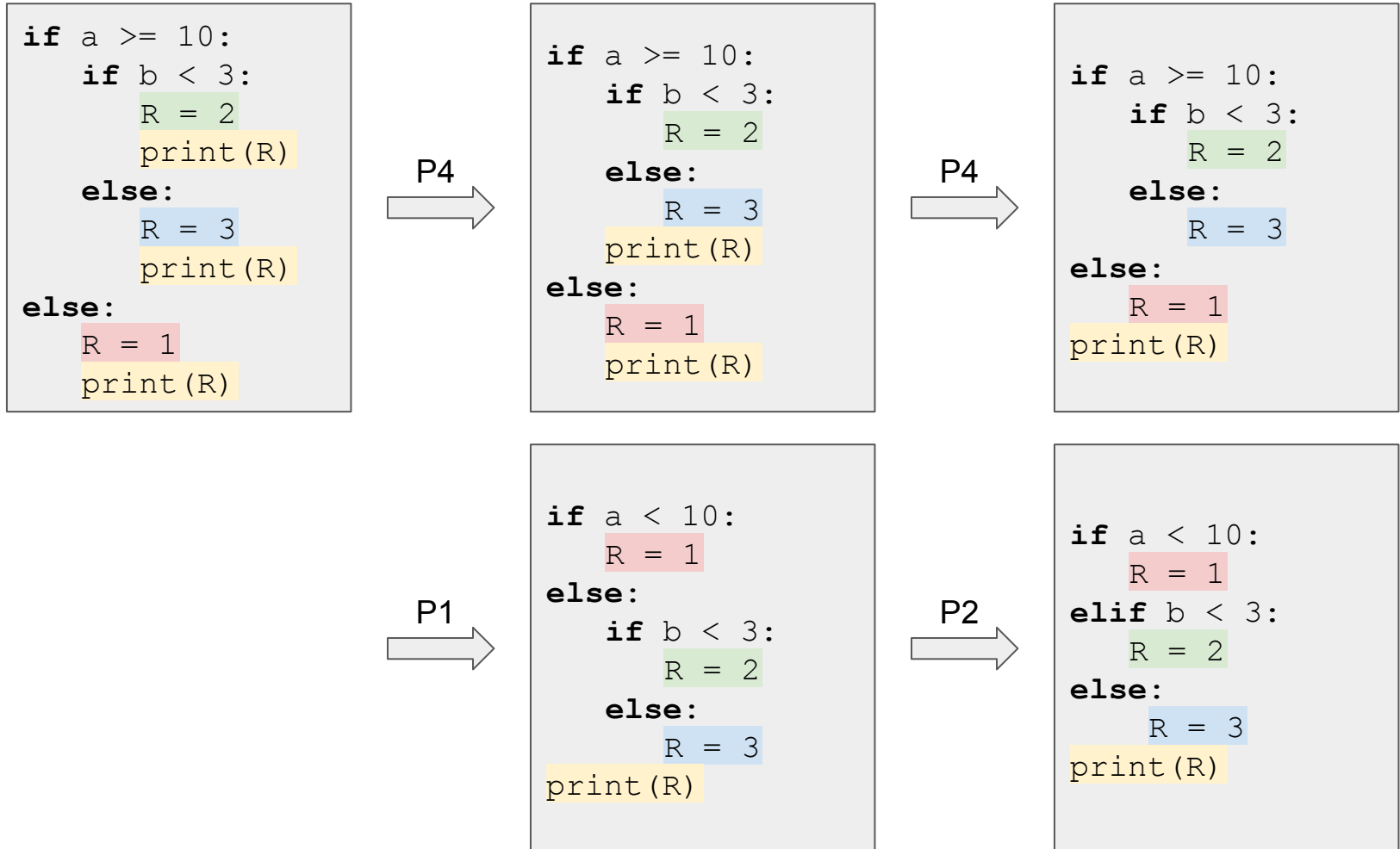
(If C1, C2 have no side effects.)



(If C1, C2 have no side effects.)

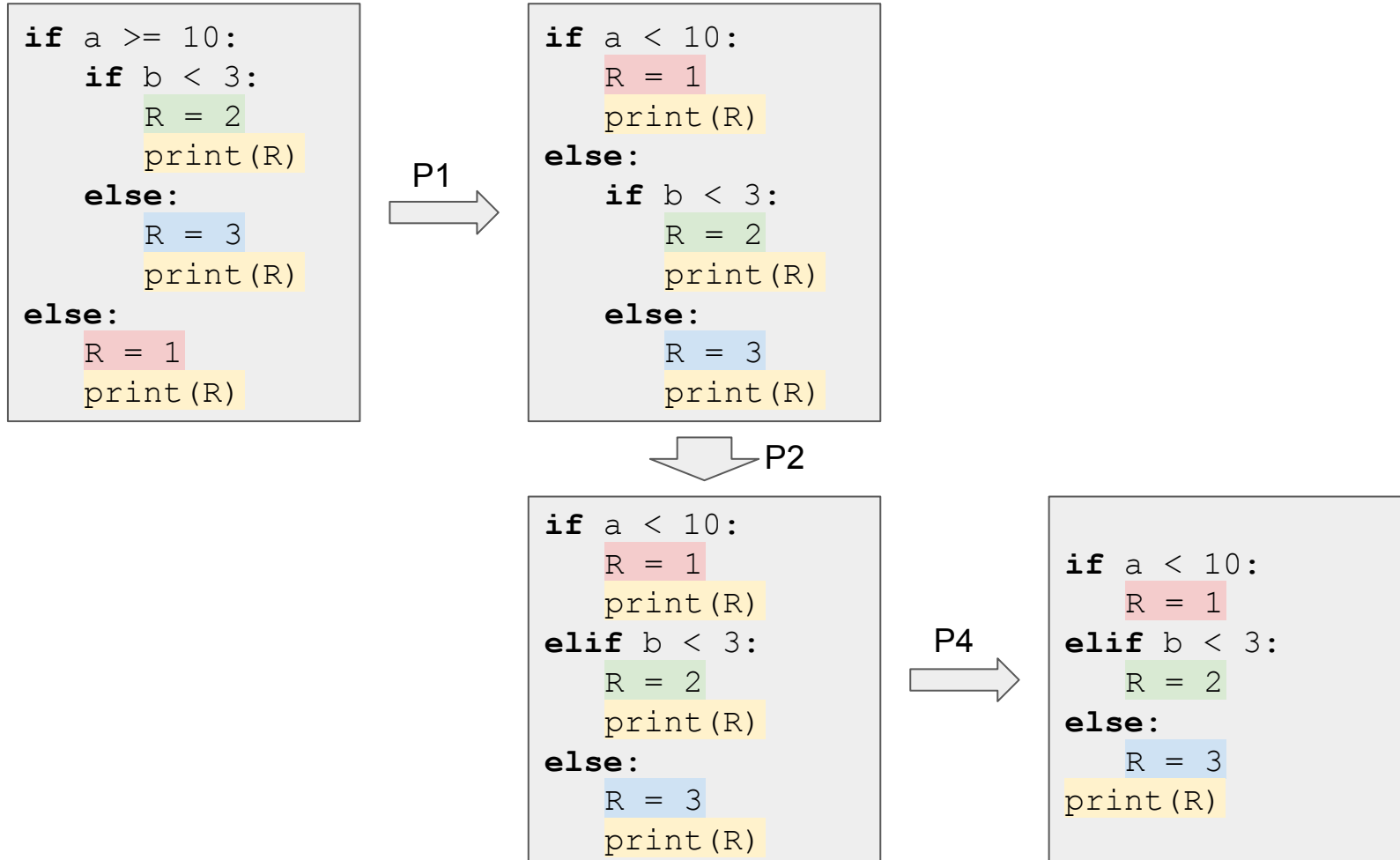
# Example: code simplification

- Applying equivalence properties may simplify the code.



# Example: code simplification

- Applying equivalence properties may simplify the code.





# Conditional expression

- Python also includes a conditional expression, based on a ternary operator:

```
expression1 if condition else expression2
```

- Uses keywords `if` and `else`, but it is an *expression*!
- The condition is evaluated first.
- If true, then `expression1` is evaluated and is the result.
- If false, then `expression2` is evaluated and is the result.

```
n = int(input("number? "))  
msg = "odd" if n%2!=0 else "even"  
print(n, "is", msg)
```

# Exercises

- [Review exercises](#) (= aula01 ex 1)

