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
```

- How do you check if X is greater than Y and Z?

a) `X > Y and Z`   b) `X > Y and X > Z`   c) `Y < X > Z`

Properties

- Remember these properties:

```
x == y   <=>   not (x != y)   <=>   y == x
x != y   <=>   not (x == y)   <=>   y != x
x > y    <=>   not (x <= y)   <=>   y < x
x <= y   <=>   not (x > y)    <=>   y >= x
```

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

```
not (not A)      <=>   A
not (A and B)    <=>   (not A) or (not B)
not (A or B)     <=>   (not A) and (not B)
A or B           <=>   B or A
A and B          <=>   B and A
A or (B and C)   <=>   (A or B) and (A or C)
A and (B or C)   <=>   (A and B) or (A and C)
```

Precedence rules

- Arithmetic > relational > `not` > `and` > `or`.

$$\texttt{x<=1+2*y**3 or n!=0 and not 1/n<=y}$$

$$\texttt{(}\underline{\texttt{x<=1+2*y**3}}\texttt{) or (}\underline{\texttt{n!=0 and not 1/n<=y}}\texttt{)}$$

$$\texttt{(x<=(}\underline{\texttt{1+2*y**3}}\texttt{)) or ((}\underline{\texttt{n!=0}}\texttt{) and (}\underline{\texttt{not 1/n<=y}}\texttt{))}$$

$$\texttt{(x<=(1+(}\underline{\texttt{2*y**3}}\texttt{))) or ((n!=0) and (not (}\underline{\texttt{1/n<=y}}\texttt{)))}$$

$$\texttt{(x<=(1+(2*(}\underline{\texttt{y**3}}\texttt{)))) or ((n!=0) and (not ((}\underline{\texttt{1/n}}\texttt{)<=y)))}$$

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:

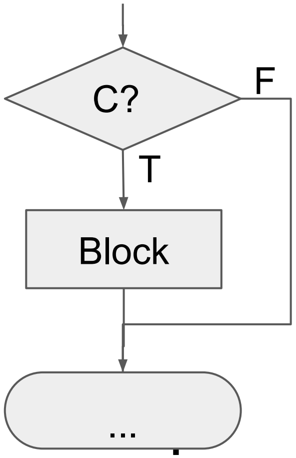
```
1/n>2 and n!=0    # if n==0, ZeroDivisionError
n!=0 and 1/n>2    # if n==0, False (1/n not evaluated)

n==0 or 3/n<4     # if n==0: True (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:

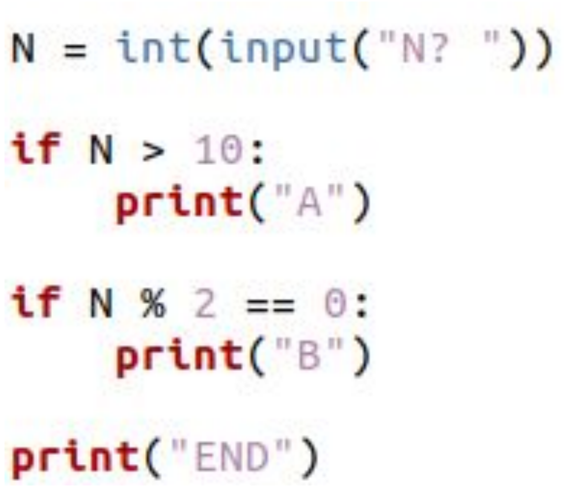
```
if condition:
    block_of_statements
...
```



- The *condition* should be a boolean expression. (Actually, it may be of any type, as it is implicitly converted to `bool`, but this could be 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

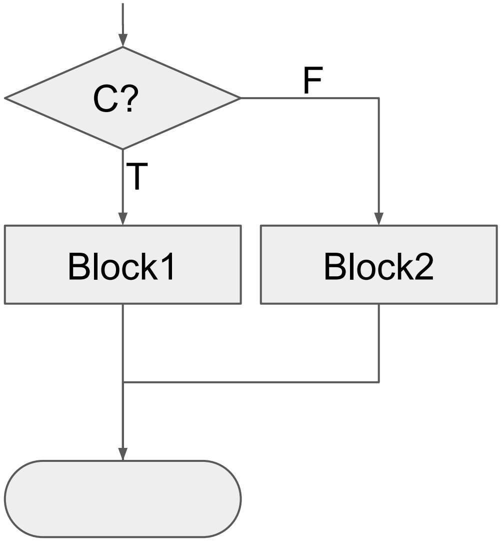
Edit Responses

Conditional execution 2: **if - else**

- A second form of the `if` statement allows selection between two alternative paths. The condition determines which one gets executed.

```
x = 3

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

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

Play ▸

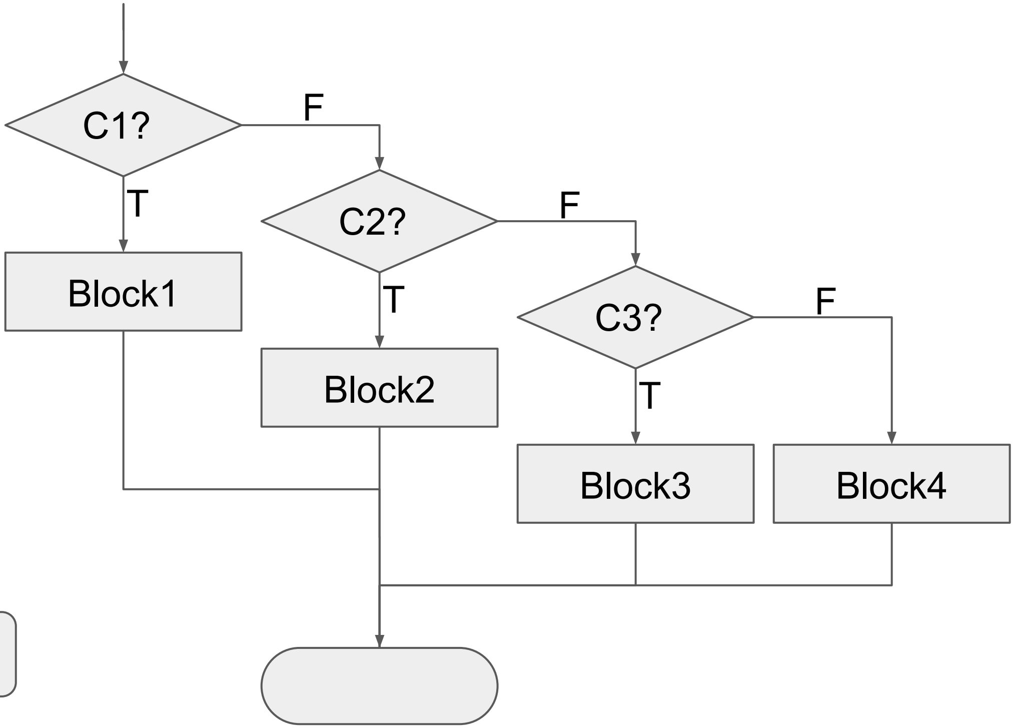


Conditional execution 3: **if - elif - else**

- 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)
```

Play ▸



Conditional statement semantics

- Which conditions select each block of statements?

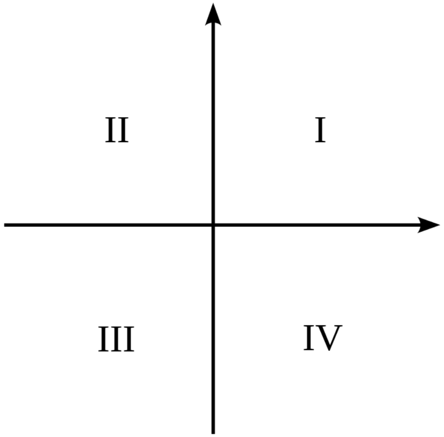
```
if C1:
    Block1        ← Block1 is executed iff C1
elif C2:
    Block2        ← Block2 is executed iff ¬C1∧C2
elif C3:
    Block3        ← Block3 is executed iff ¬C1∧¬C2∧C3
else:
    Block4        ← Block4 is executed iff ¬C1∧¬C2∧¬C3

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 C:
    BlockX
else:
    BlockY
```

P1 ≡

```
if not C:
    BlockY
else:
    BlockX
```

```
if C1:
    BlockX
else:
    if C2:
        BlockY
    else:
        BlockZ
```

P2 ≡

```
if C1:
    BlockX
elif C2:
    BlockY
else:
    BlockZ
```

```
if C1:
    BlockA
    BlockX
elif C2:
    BlockA
    BlockY
else:
    BlockA
    BlockZ
```

P3 ≡

```
BlockA
if C1:
    BlockX
elif C2:
    BlockY
else:
    BlockZ
```

(If C1, C2 have no side effects.)

```
if C1:
    BlockX
    BlockB
elif C2:
    BlockY
    BlockB
else:
    BlockZ
    BlockB
```

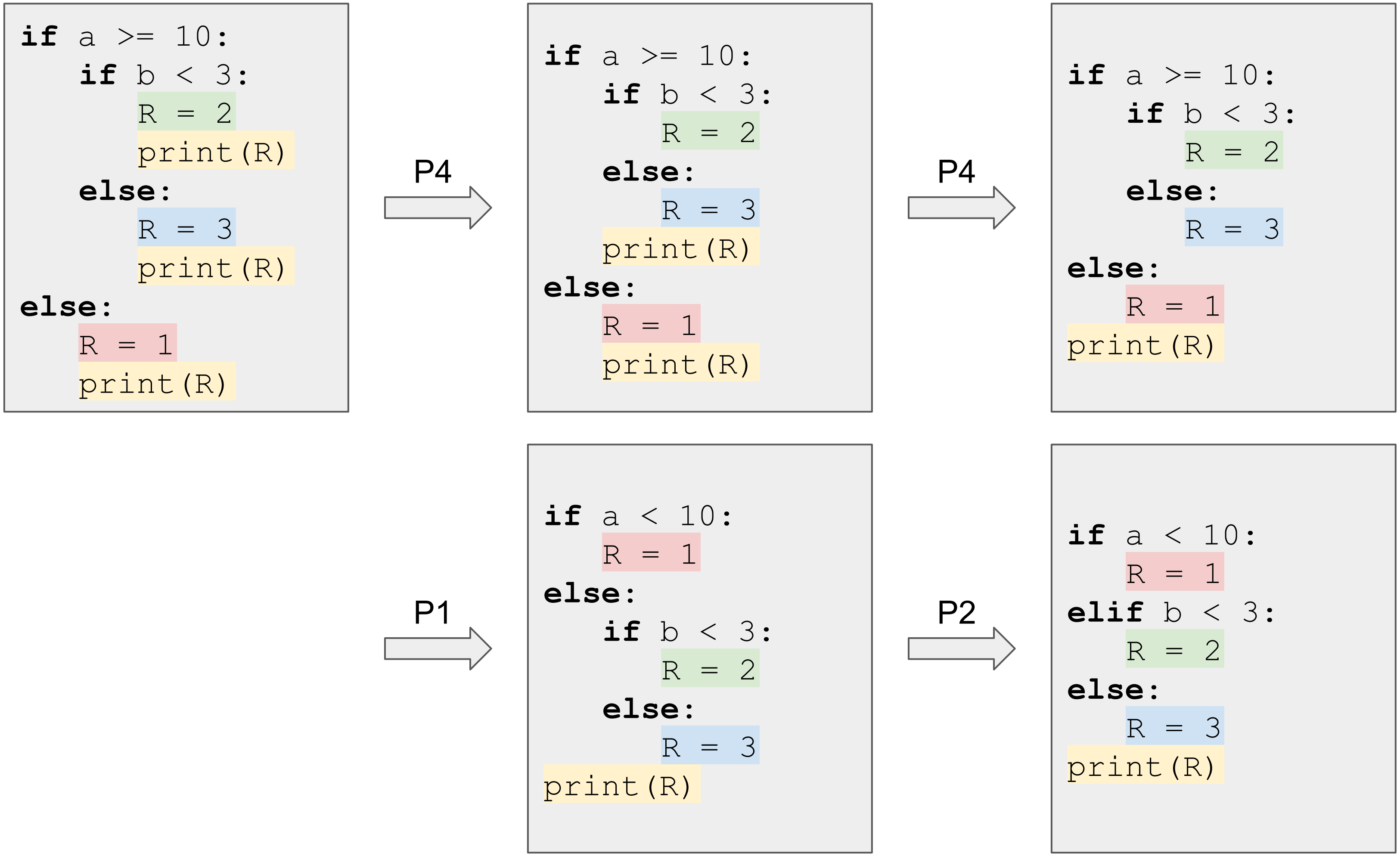
P4 ≡

```
if C1:
    BlockX
elif C2:
    BlockY
else:
    BlockZ
BlockB
```

(If C1, C2 have no side effects.)

Example: code simplification

- Applying equivalence properties may simplify the code.



```
if a >= 10:
    if b < 3:
        R = 2
        print(R)
    else:
        R = 3
        print(R)
else:
    R = 1
    print(R)
```

```
if a >= 10:
    if b < 3:
        R = 2
    else:
        R = 3
    print(R)
else:
    R = 1
    print(R)
```

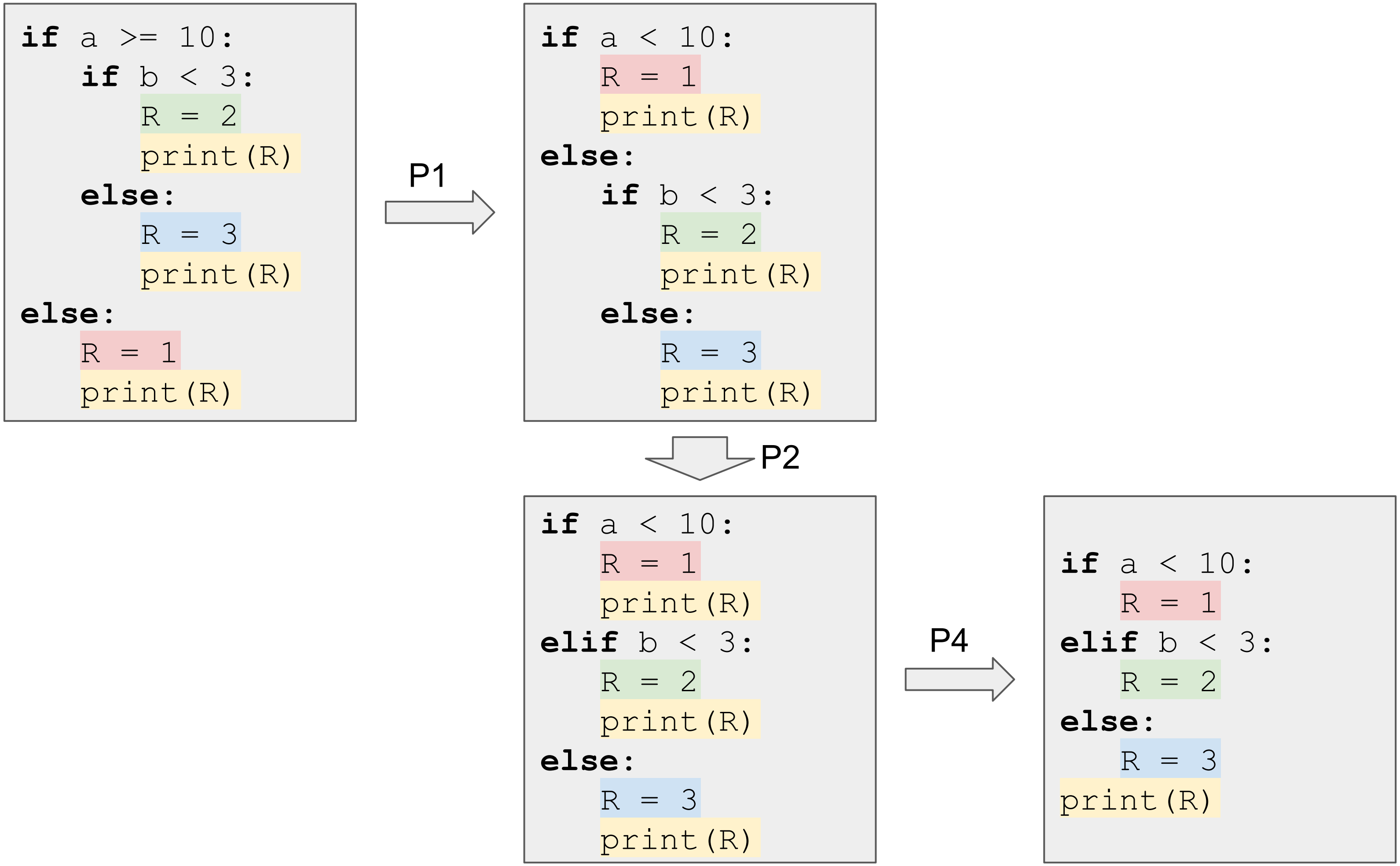
```
if a >= 10:
    if b < 3:
        R = 2
    else:
        R = 3
else:
    R = 1
print(R)
```

```
if a < 10:
    R = 1
else:
    if b < 3:
        R = 2
    else:
        R = 3
print(R)
```

```
if a < 10:
    R = 1
elif b < 3:
    R = 2
else:
    R = 3
print(R)
```

Example: code simplification

- Applying equivalence properties may simplify the code.



```
if a >= 10:
    if b < 3:
        R = 2
        print(R)
    else:
        R = 3
        print(R)
else:
    R = 1
    print(R)
```

```
if a < 10:
    R = 1
    print(R)
else:
    if b < 3:
        R = 2
        print(R)
    else:
        R = 3
        print(R)
```

```
if a < 10:
    R = 1
    print(R)
elif b < 3:
    R = 2
    print(R)
else:
    R = 3
    print(R)
```

```
if a < 10:
    R = 1
elif b < 3:
    R = 2
else:
    R = 3
print(R)
```

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 (= aula02 ex 1)