



deti

universidade de aveiro  
departamento de electrónica,  
telecomunicações e informática

# Fundamentos de Programação

António J. R. Neves  
João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática  
Universidade de Aveiro

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

- 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 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
```

- Remember these properties:

$x == y \iff \text{not } x != y \iff y == x$

$x != y \iff \text{not } x == y \iff y != x$

$x > y \iff \text{not } x \leq y \iff y < x$

$x \leq y \iff \text{not } x > y \iff y \geq x$

$\text{not } (\text{not } A) \iff A$

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

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

- And these (but beware of *short-circuit evaluation*\*):

$A \text{ or } B \iff B \text{ or } A$

$A \text{ and } B \iff B \text{ and } A$

$A \text{ or } (B \text{ and } C) \iff (A \text{ or } B) \text{ and } (A \text{ or } C)$

$A \text{ and } (B \text{ or } C) \iff (A \text{ and } B) \text{ or } (A \text{ and } C)$

- Arithmetic > relational > not > and > or.

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

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

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

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

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

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

`X and Y`      # if X is false then X, otherwise Y

`X or Y`      # if X is true then X, otherwise Y

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

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

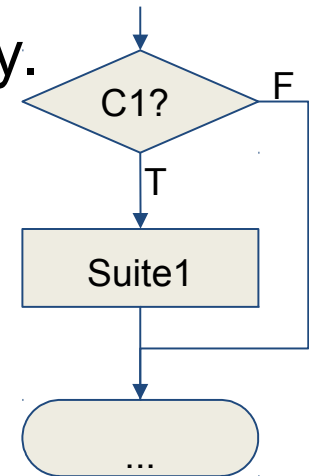
`n!=0 and 1/n>2`      # False if `n==0`, `1/n` not evaluated

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

- But remember:
  - Commutative and distributive properties may not be valid!

- **Conditional statements** allow the program to check conditions and change its behavior accordingly.
- The simplest form is the `if` statement:

```
if condition:  
    statements  
...
```



- The *condition* is an expression of any type, but it is implicitly converted to `bool`. (**Warning:** this may be **surprising!** Please use proper boolean expressions.)
- The indented *suite of statements* gets executed if the condition is true. If not, execution continues after the indented statements.
- The suite must have one or more statements.



# Conditional execution (2)

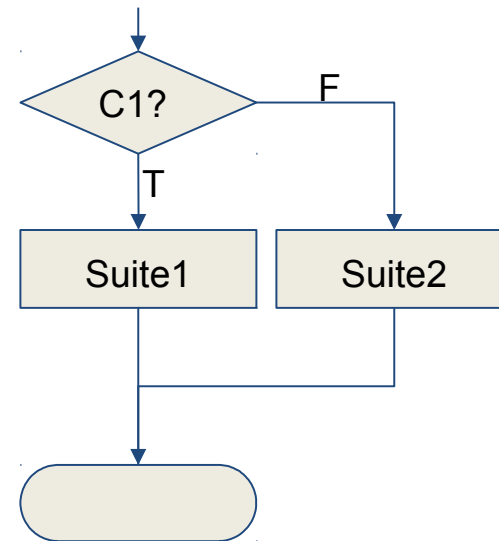


deti

universidade de aveiro  
departamento de electrónica,  
telecomunicações e informática

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

```
if x%2 == 0:  
    print('x is even')  
else:  
    print('x is odd')  
  
#END
```



# Conditional execution (3)

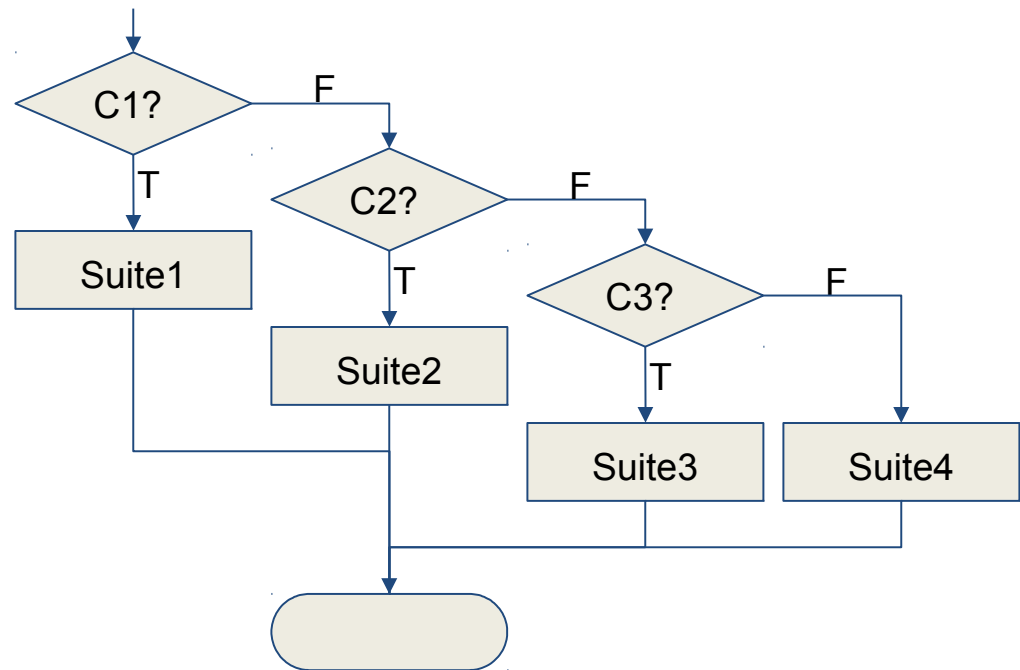


deti

universidade de aveiro  
departamento de electrónica,  
telecomunicações e informática

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

```
if x < 10:  
    mark = 'Poor'  
elif x < 13:  
    mark = 'Reasonable'  
elif x < 17:  
    mark = 'Good'  
else:  
    mark = 'Excelent'  
  
print(mark)
```



# Conditional execution (4)



deti

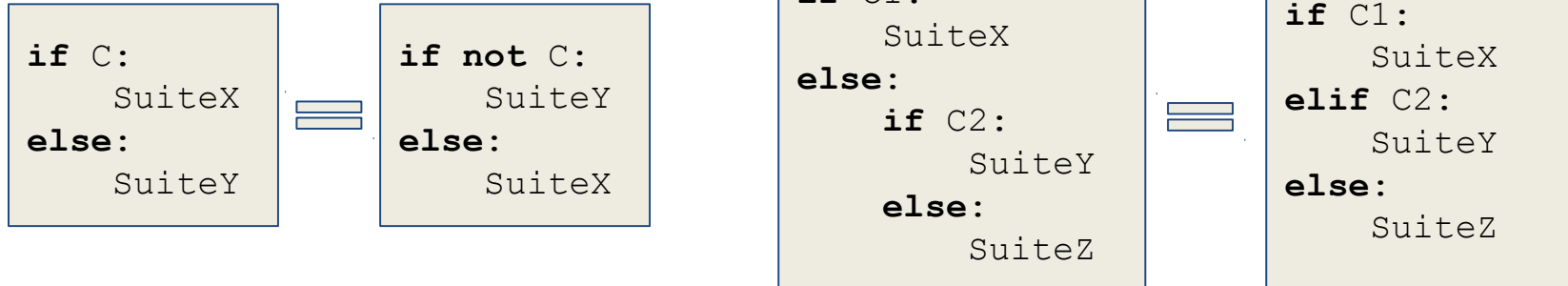
universidade de aveiro  
departamento de electrónica,  
telecomunicações e informática

- One conditional can also be nested within another.

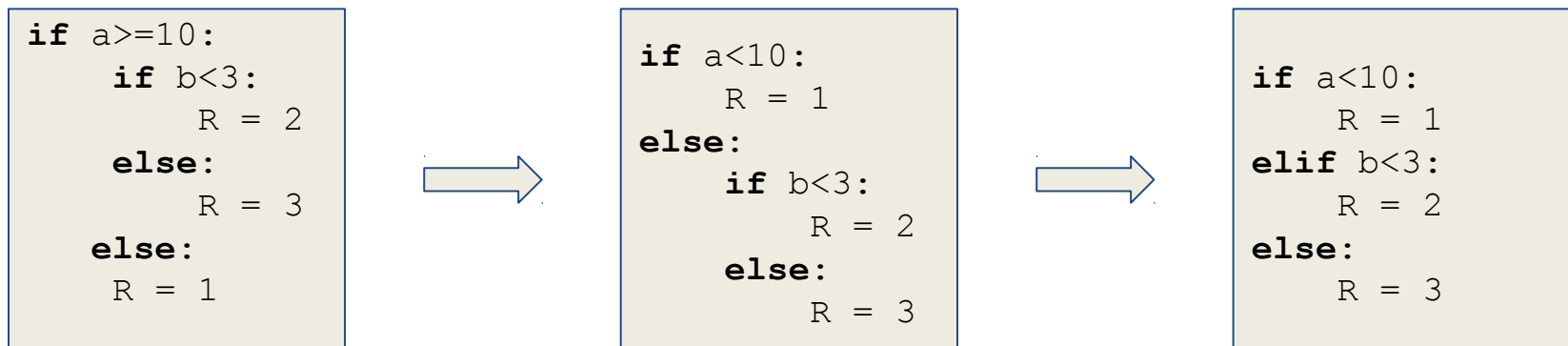
```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

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

- Code equivalence properties.



- Transformations may simplify the code.



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

```
expression1 if condition else expression2
```

- Uses keywords **if** and **else**, but it's not a statement!
- 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)
```