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

deti · universidade de aveiro
departamento de electrónica,
telecomunicações e informática

- **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    <=>    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
not (not A)       <=>    A
not (A and B)    <=>     (not A) or (not B)
not (A or B)     <=>     (not A) and (not B)
```

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

```
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`.

```
        x<=1+2*y**3 or n!=0 and not 1/n<=y

      (x<=1+2*y**3) or (n!=0 and not 1/n<=y)

     (x<=(1+2*y**3)) or ((n!=0) and (not 1/n<=y))

    (x<=(1+(2*y**3))) or ((n!=0) and (not (1/n<=y)))

  (x<=(1+(2*(y**3)))) or ((n!=0) and (not ((1/n)<=y)))
```

Short-circuit evaluation

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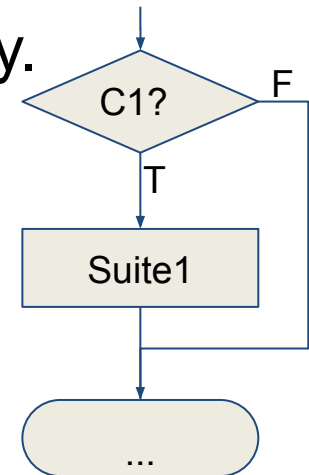
- But remember:
  - Commutative and distributive properties may not be valid!

- **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 <u>implicitly converted</u> 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.

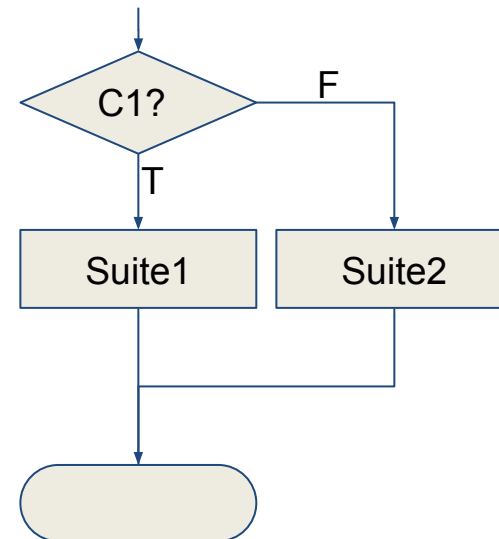- A second form of the `if` statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed.

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

#END
```

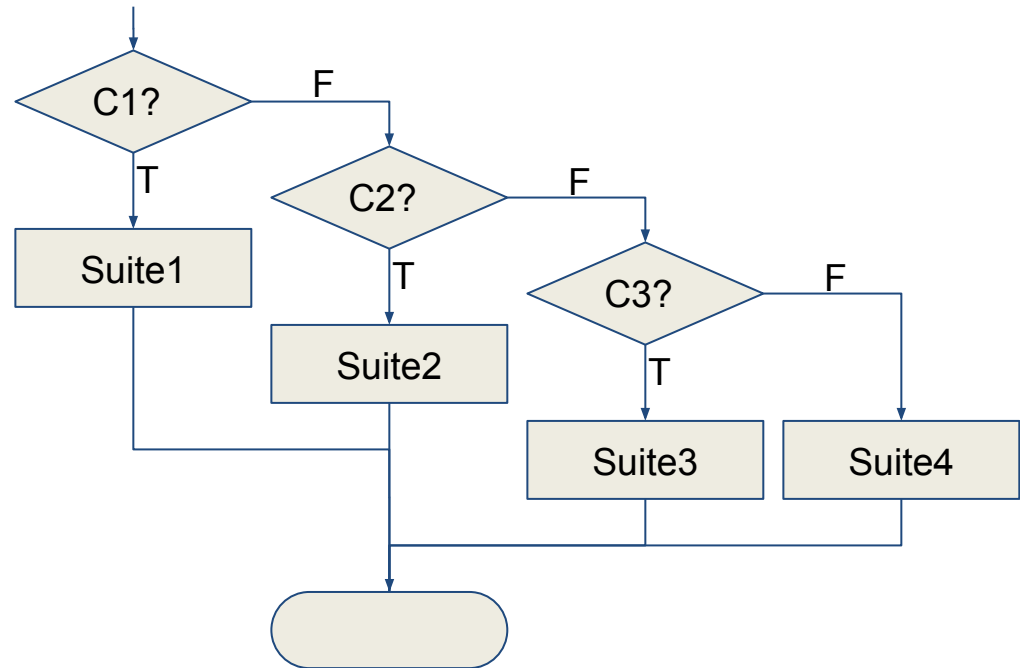- Sometimes there are more than two possibilities and we need more than two branches (chained conditional).

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

print(mark)
```

Conditional execution (4)

universidade de aveiro
departamento de electrónica,
telecomunicações e informática

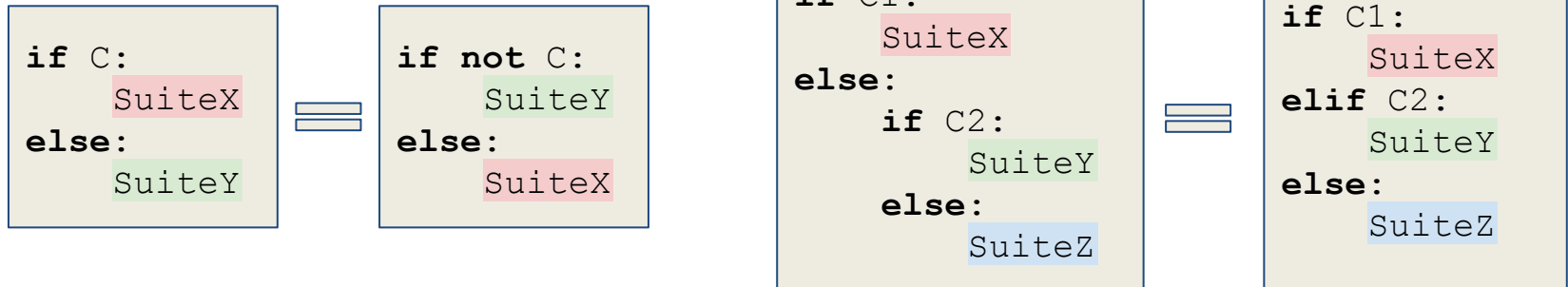- One conditional can also be nested within another.
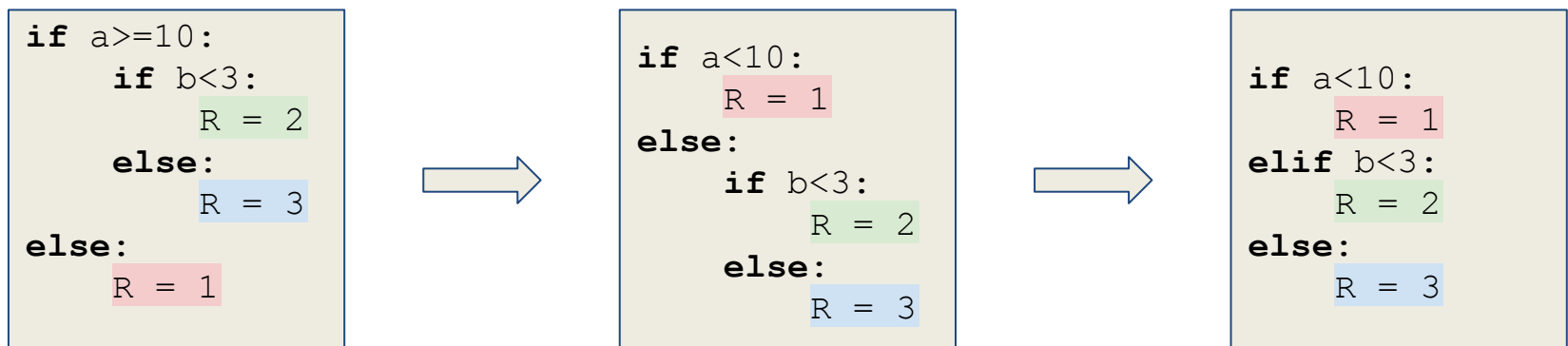
```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

- 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 transformations

- Code equivalence properties.

```
if C:
    SuiteX
else:
    SuiteY
```
=
```
if not C:
    SuiteY
else:
    SuiteX
```

```
if C1:
    SuiteX
else:
    if C2:
        SuiteY
    else:
        SuiteZ
```
=
```
if C1:
    SuiteX
elif C2:
    SuiteY
else:
    SuiteZ
```

- Transformations may simplify the code.

```
if a>=10:
    if b<3:
        R = 2
    else:
        R = 3
else:
    R = 1
```
⟹
```
if a<10:
    R = 1
else:
    if b<3:
        R = 2
    else:
        R = 3
```
⟹
```
if a<10:
    R = 1
elif b<3:
    R = 2
else:
    R = 3
```

Conditional expression

deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

- 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)
```