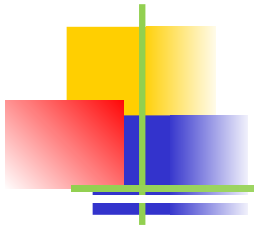


Fundamentos de Programação

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

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

- 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    <=>    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`.

```
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!

```
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:

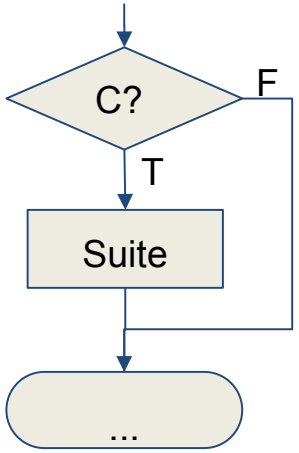
```
1/n>2 and n!=0    # ZeroDivisionError if n==0
n!=0 and 1/n>2    # False if n==0, 1/n not evaluated
n==0 or 3/n<4     # True if n==0, 3/n not evaluated
```

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

Conditional execution (1)

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

```
if condition:
    suite_of_statements
...
```



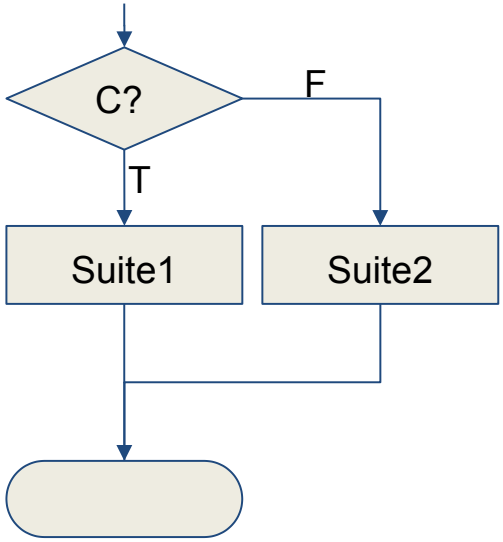
- The *condition* should be a boolean expression.
  - (It may be of any type, and is implicitly converted to `bool`, but this is confusing and should be avoided.)
- The *suite of statements* is executed if the condition is true. If not, execution continues after those statements.
- The suite must have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

Conditional execution (2)

- 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.

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

#END
```

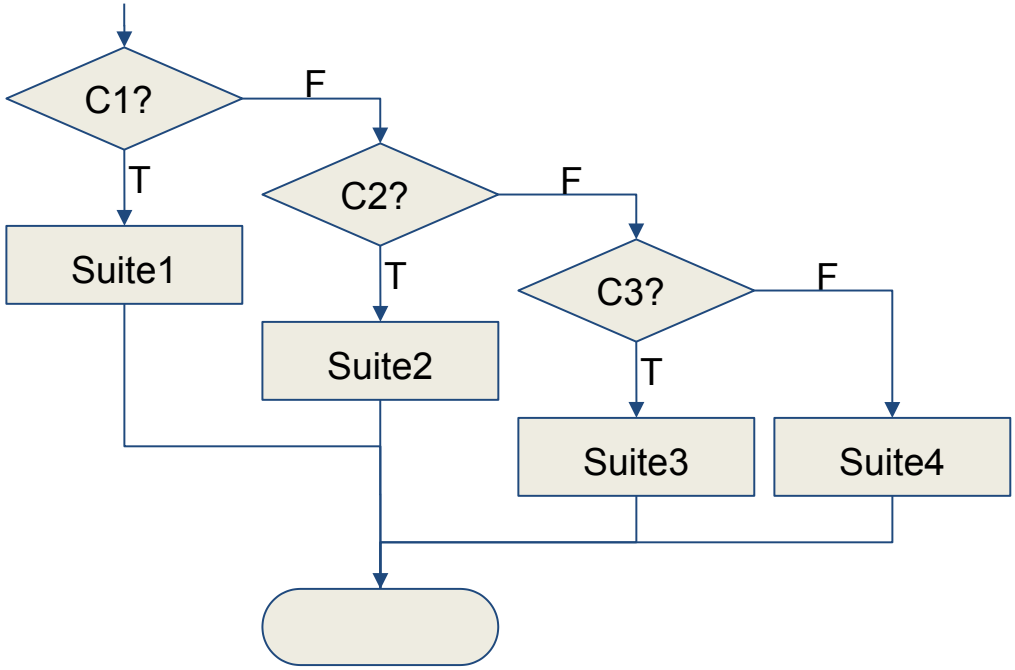


Conditional execution (3)

- Sometimes there are more than two alternatives 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 statement semantics

- Which conditions select each suite of statements?

```
if C1:
    Suite1      ← Suite1 is executed iff C1
elif C2:
    Suite2      ← Suite2 is executed iff ¬C1∧C2
elif C3:
    Suite3      ← Suite3 is executed iff ¬C1∧¬C2∧C3
else:
    Suite4      ← Suite4 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
```

- 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

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

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

```
if C:
    SuiteX
else:
    SuiteY
```

P1 =

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

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

P2 =

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

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

P3 =

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

(If C1, C2 have no side effects.)

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

P4 =

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

(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)
```

P4

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

P4

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

P1

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

P2

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

Example: code simplification

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

- 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)
```

P1

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

P2

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

P4

```
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)
```