COSC2429 Introduction to Programming

Selections

Quang Tran <quang.tran@rmit.edu.vn>

Outline

- Boolean values and Boolean expressions
- Comparison operators
- Logical operators
- Precedence of operators
- Summary of operator precedence
- Conditional execution: binary selection
- Omitting the else clause: unary selection
- Nested conditionals
- Chained conditionals
- Boolean functions
- **break** and **continue** statement
- Exercise: Final grade

Boolean values and Boolean expressions

- The Boolean data type **bool** in Python (named after the British mathematician George Boole) has only 2 values: **True** and **False**

```
print(True)
print(type(True))
print(type(False))
```

- A **Boolean expression** is an expression that evaluates to a Boolean value
- The equality operator == compares two values and produces a Boolean value related to whether the two values are equal to one another

```
print(5 == 5)
print(5 == 6)
```

Comparison operators

- The == operator is one of six common **comparison operators**:

```
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
```

- A common error is to use a single equal sign (=) instead of a double equal sign (==)
- Remember that = is an assignment operator and == is a comparison operator
- Also, there is no such thing as =< or =>

Logical operators

- There are three **logical operators**: **and**, **or**, **not**
- The semantics (meaning) of these operators is similar to their meaning in English

```
x = 5
print(x > 0 and x <10)

n = 25
print(n % 2 == 0 or n % 3 == 0)
```

Precedence of operators

- Python will always evaluate the arithmetic operators first (exponentiation is highest, then multiplication/division, then addition/subtraction)
- Next comes the comparison operators (sometime called relational operators)
- Finally, the logical operators are done last
- Consider this example:

```
x * 5 >= 10 and y – 6 <= 20
```

- Although many programmers might place parenthesis around the two comparison expressions as below, it is not necessary due to the precedence of operators

```
(x * 5 >= 10) and (y – 6 <= 20)
```

Summary of operator precedence

Level	Category	Operators
7	exponentiation	**
6	multiplication	*, /, //, %
5	addition	+, -
4	comparison	==, !=, <=, >=, >, <
3	logical	not
2	logical	and
1	logical	or

Conditional execution: binary selection

- In order to write useful programs, we almost always need the ability to check conditions and change the behaviour of the program accordingly.
- **Selection statements** (also know as **conditional statements**) give us this ability
- The simplest form of selection is the **if statement**. It is sometimes referred to as **binary selection** since there are two possible paths of execution

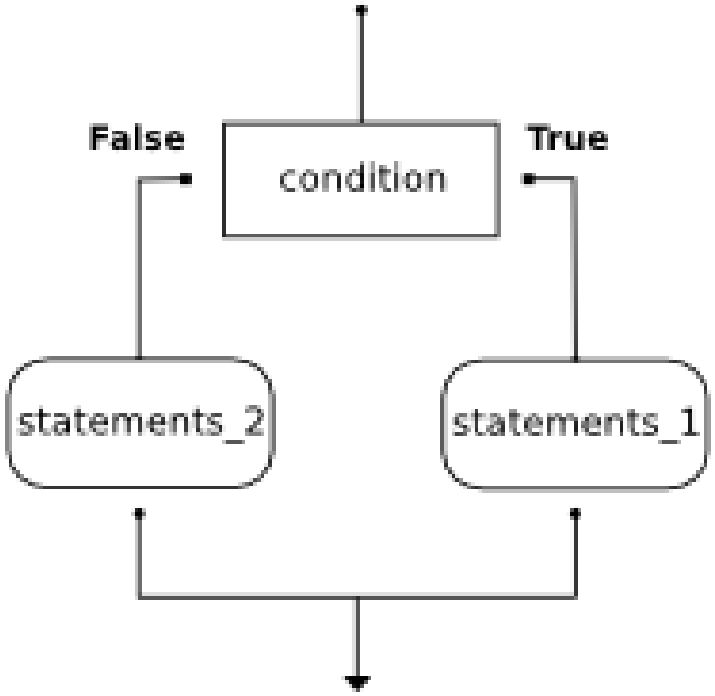
```
x = int(input("Please enter an integer: "))

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

Conditional execution: binary selection

- The syntax for an if statement looks like this:

```
if boolean expression:
    statements_1          # executed if condition evaluates to True
else:
    statements_2          # executed if condition evaluates to False
```

- The indented statements that follow are called a **block**. The first unindented statement marks the end of the block.
- There is no limit on the number of statements that can appear under the two clauses of an if statement, but there has to be at least one statement in each block.

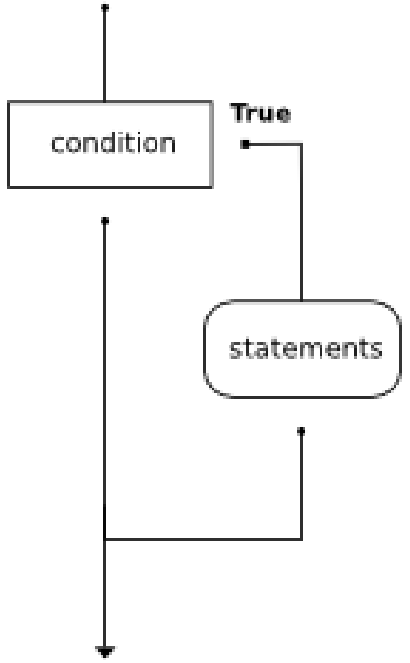


Omitting the else clause: unary selection

- Another form of the if statement is one in which the else clause is omitted entirely. This creates what is sometimes called **unary selection**.

```
x = int(input("Please enter an integer: "))

if x < 0:
    print("The negative number ",  x, " is not valid here. ")
print("This is always printed")
```

- In this case, when the condition evaluates to **True**, the statements are executed.
- Otherwise the flow of execution continues to the statement after the body of the if.



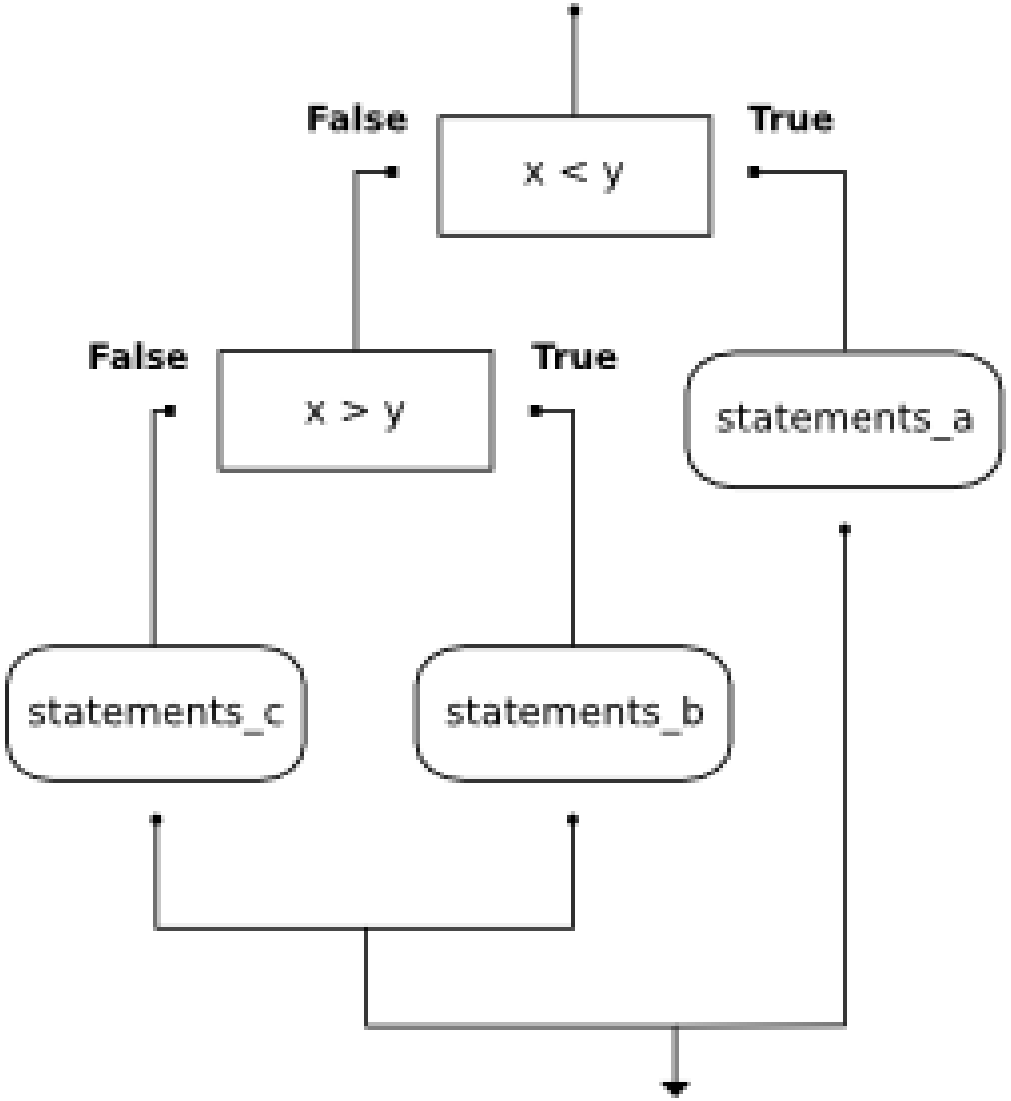
Nested conditionals

- One **conditional** can be nested within another.
- For example, assume we have two integer variables, x and y.
- The following pattern of selection show how we might decide how are they related to each other.

```
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```

Nested conditionals

- The flow of control for this example can be seen in this flowchart illustration.



Chained conditionals

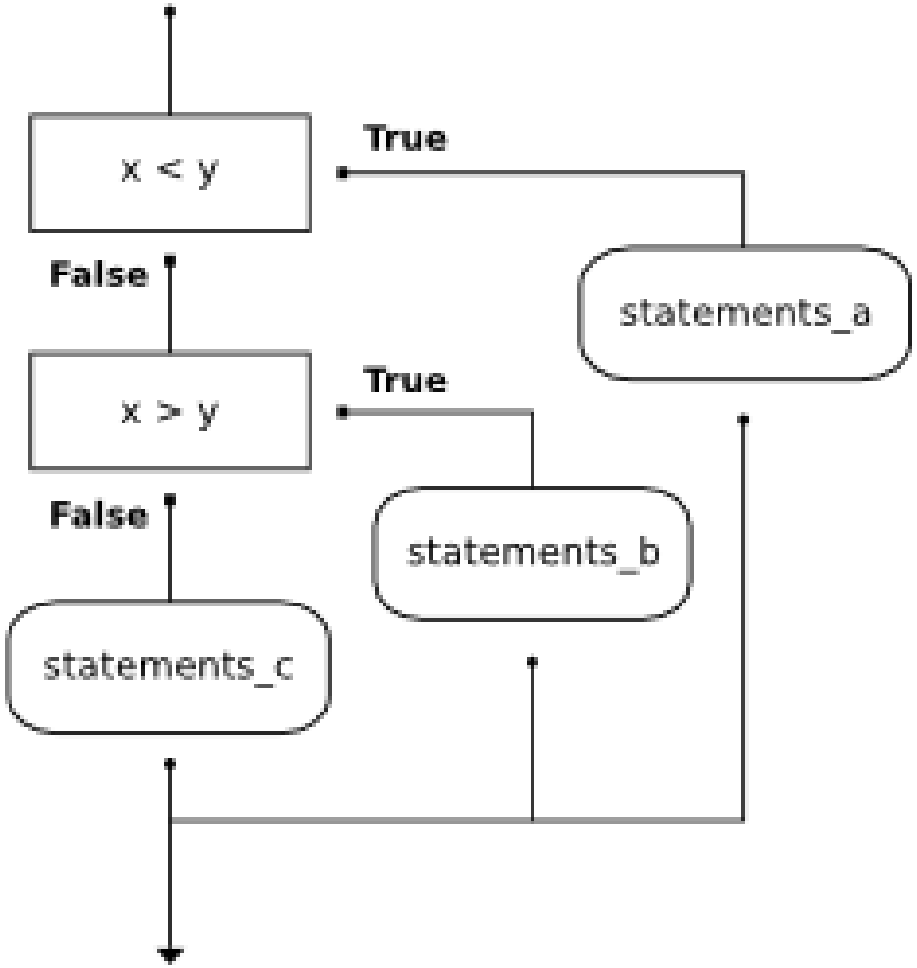
- Python provides an alternative and preferred way to write nested selection such as the one shown in the previous section.
- This is sometimes referred to as a **chained conditional**

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```

- Exactly one branch will be executed.
- There is no limit of the number of **elif** statements but only a single (and optional) final **else** statement is allowed and it must be the last branch in the statement.

Chained conditionals

- The flow of control can be drawn in a different orientation but the resulting pattern is identical to the one shown above.



Boolean function

- We have already seen that Boolean values result from the evaluation of Boolean expressions.
- Since the result of any expression evaluation can be returned by a function (using the **return** statement), functions can return Boolean values.
- This turns out to be a very convenient way to hide the details of complicated tests.
- Let’s look at the following example.

Boolean function

```
def is_divisible(x, y):
    """
        Check if integer x is divisible by integer y
        :param x: an integer
        :param y: another integer
        :return: True if x is divisible by y and False otherwise
    """
    if x % y == 0:
        result = True
    else:
        result = False

    return result

if is_divisible(10, 5):
    print("10 is divisible by 5")
else:
    print("10 is not divisible by 5")
```

Boolean function

- We can make the function more concise by taking advantage of the fact that the condition of the if statement is itself a Boolean expression.
- We can return it directly, avoiding the if statement altogether:

```
def is_divisible(x, y):
    """
        Check if integer x is divisible by integer y
        :param x: an integer
        :param y: another integer
        :return: True if x is divisible by y and False otherwise
    """
    return x % y == 0

if is_divisible(10, 5):
    print "10 is divisible by 5"
else:
    print "10 is not divisible by 5"
```

break statement

- **break** statement is used to quit a loop entirely
- What will be printed out in the following code fragments?

```
for i in range(1000):
    square = i * i
    if square >= 100:
        break
    print(i)
```

```
i = 0
while i < 1000:
    square = i * i
    if square >= 100:
        break
    print(i)
    i += 1
```

continue statement

- **continue** statement is used to skip the current iteration of a loop and run the next iteration
- What will be printed out in the following code fragments?

```
for i in range(10):
    square = i * i
    if square == 25:
        continue
    print(i)
```

```
i = 0
while i < 10:
    square = i * i
    if square == 25:
        continue
    print(i)
    i += 1
```

Example: Final grade

- Write a function which takes a final result – a number between 0 and 100 - as parameter, and returns the final grade for that result according to the following table:

Final result	Final grade
>= 80	HD
[70, 80)	DI
[60, 70)	CR
[50, 60)	PA
< 50	NN

- Test your function by writing a program that asks user for a final result and then print out the corresponding final grade