COMP 10280 Programming I (Conversion)

John Dunnion

School of Computer Science
University College Dublin

COMP 10280 Programming I (Conversion)/Lecture 7

Outline

Comparison operators

Boolean operators

Conditional statement

Conditional statement in Python

Comparison operators in Python

Python Operator	**Operation**
$==$	Equals
$!=$	Not equals
$<$	Less than
$<=$	Less than or equal to
$>$	Greater than
$>=$	Greater than or equal to

Boolean operators in Python 3.x

- There are three Boolean operators: `and`, `or` and `not`
- `a and b`: If `a` is `False`, it returns `a`, otherwise it returns `b`
- `a or b`: If `a` is `False`, it returns `b`, otherwise it returns `a`
- `not a`: If `a` is `False`, it returns `True`, otherwise it returns `False`

Python Operator	**Operation**
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

Boolean operators in Python 2.x

- There are three Boolean operators: `and`, `or` and `not`
- `a and b` is `True` if both `a` and `b` are `True`, otherwise it is `False`
- `a or b` is `True` if one of `a` and `b` is `True` (or both are `True`), otherwise it is `False`
- `not a` is `False` if `a` is `True`, and `True` if `a` is `False`

Python Operator	**Operation**
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

Using Boolean operators in Python

```
>>> a = 2
>>> b = 3
>>> c = 10
>>> d = 10
>>> a < b
True
>>> c > b
True
>>> c < d
False
>>> d == d
True
>>> c == d
True
>>> c != d
False
```

Conditions

- We are familiar with making decisions based on conditions
- *If I am hungry, I will eat my dinner*
- *If I am cold, I will put on my coat*
- *If the number is even, I will divide the number by 2*
- Such sentences are called conditional sentences
- Such sentences have two parts:
  - A condition or test:
    If I am hungry, *If I am cold*, *If the number is even*
  - An action:
    I will eat my dinner, *I will put on my coat*, *I will divide the number by 2*
- The action will only be carried out if the condition is satisfied (or the test is true)
- Optionally, there is another action that will be carried out if the condition is not satisfied (or the test is false)

Sequential statements

- The programs that we have seen so far have contained only sequential statements
- Such programs follow a sequential flow of control
- There is a single execution path through the program
- These can be called *straight-line programs*
- In such a program, statements are executed in the order in which they appear
- The program stops when control reaches the final statement
- The type of problem that we can solve with such a program is very simple and very limited

Conditional statements

- Most programming languages allow for programs that have more than one execution path through them
- Such programs follow a conditional flow of control
- These can be called *branching programs*
- A conditional statement has two or three parts:
    - Optionally, a statement, or block of statements, that is executed when the condition evaluates to `False`
    - A test, ie an expression that evaluates to either `True` or `False`
    - A statement, or block of statements, that is executed when the condition evaluates to `True`
- After the conditional statement, execution resumes at the statement following the conditional statement

Conditional statements

- Conditional statements allow us to change the *flow of control* in a program
- Within a program, a condition can be tested and actions carried out only if the condition is `True`
- This gives programs much more power and flexibility

Conditional statement in Python (1)

- In Python, a conditional statement has one of the following forms:
- **if** *Boolean expression*:

    statement(s)
- **if** *Boolean expression*:

    statement(s)

  else:

    statement(s)
- **if** *Boolean expression*:

    statement(s)

  elif *Boolean expression*:

    statement(s)

  else:

    statement(s)

Conditional statement in Python (2)

- In describing the forms of the conditional statement, italics are used to describe the type of Python code that can occur at that point in the statement
- *Boolean expression* indicates that any expression that evaluates to `True` or `False` can follow the reserved words `if` or `elif`
- *statement(s)* indicates that any sequence of Python statements can appear at those points

Using the onditional statement in Python (1)

- Consider the following program that prints “Number is zero” if the number entered by the user is 0

```
# Using the conditional statement
# Prints 'Number is zero' if the number
        entered is 0
# p25.py

# Ask the user for input
# Use a cast to make it an int
number = int(input('Enter␣an␣int:␣'))

if number == 0:
    print('Number␣is␣zero')
print('Finished!')
```

Using the onditional statement in Python (2)

- Example outputs from this program are the following:

```
>>>
Enter an int: 123
Finished!
>>>
>>>
Enter an int: 0
Number is zero
Finished!
>>>
>>>
Enter an int: -5
Finished!
>>>
```

Using the onditional statement in Python (3)

- Consider the following program that tests the number entered by the user and prints “Number is even” or “Number is odd”

```
# Using the conditional statement
# Prints 'Number is even' or
# 'Number is odd'
# p26.py

# Ask the user for input
# Use a cast to make it an int
number = int(input('Enter␣an␣int:␣'))

if number % 2 == 0:
    print('Number␣is␣even')
else:
    print('Number␣is␣odd')
print('Finished!')
```

Using the onditional statement in Python (4)

- Example outputs from this program are the following:

```
>>>
Enter an int: 3
Number is odd
Finished!
>>>
>>>
Enter an int: 2424
Number is even
Finished!
>>>
```

Evaluating the Boolean expression

- The expression `number % 2 == 0` evaluates to `True` when the remainder of `number` divided by 2 is 0, and evaluates to `False` otherwise
- Recall that == is the operator used for comparison
- The = operator is used only for assignment

Indentation

- Note that indentation is semantically significant in Python
- Statements at the same level of indentation belong to the same block of statements
- Different languages use different mechanisms to mark blocks of statements
- For example, Pascal uses `begin` and `end` keywords
- C and Java use braces (curly brackets), ie `{` and `}`
- Some languages use the keyword that introduces the block spelled backwards, eg `if` and `fi`
- Python is unusual in using indentation in this way.
- “The off-side rule”
- Programs *should* be indented
- Python’s indentation forces the programmer to indent their programs properly and in a standard way

Currency Conversion Program: Algorithm

- Consider a more sophisticated program to convert Euro to Dollars
- We only want to convert Euro amounts that are greater than zero
- We start off by writing an algorithm for this program

```
Prompt the user for a Euro amount
Read the Euro amount
if the Euro amount $\geq$ 0 then
    Perform the conversion
    Print out the Dollar amount
else
    Tell the user that the amount must be $\geq$ 0
Program finishes
```

Currency Conversion Program: Program

```
# Converting Euro to US Dollars
# p27.py

euro_dollar_conversion = 1.11740     # Number of US
                                     # According to

# Ask the user to enter the Euro amount
euro_amount = int(input('Enter␣the␣amount␣of␣Euro␣y
print('Amount␣in␣Euro:', euro_amount)

if euro_amount >= 0:
    print('Amount␣in␣US␣Dollars:', euro_amount * eu
else:
    print('Amount␣must␣be␣>=␣0.')
    print('Please␣try␣again.')

print('Finished!')
```

Currency Conversion Program: Output

- Example outputs from this program are the following:

```
Enter the amount of Euro you wish to convert: 10
Amount in Euro: 1000
Amount in US Dollars: 1117.3999999999999
Finished!
>>> ============================ RESTART ===
Enter the amount of Euro you wish to convert: 0
Amount in Euro: 0
Amount in US Dollars: 0.0
Finished!
>>> ============================ RESTART ===
Enter the amount of Euro you wish to convert: −1
Amount in Euro: −1
Amount must be >= 0.
Please try again.
Finished!
>>>
```