Introduction to: Computers & Programming: Booleans, Conditionals and Loops: Flow of Control in Python

Adam Meyers
New York University
Outline

• What is flow of control?
• Order of statements
  – Within a function
  – Functions within functions
• Boolean Data Type
• Logical Operators
• Conditional Statements
  – Conditional Keywords: if, else, elif
  – Application: simple decision Trees
Flow of Control

• The determination of when and if
  – instructions execute, functions are called, variables are set, output is returned, etc.

• Simple cases
  – Within a block, instructions execute top to bottom
    ```python
def print_three_things(thing1, thing2, thing3):
    print(thing1)
    print(thing2)
    print(thing3)
```
  – Nested blocks are also executed in order
    ```python
def print_three_things_three_times(thing1, thing2, thing3):
    print_three_things(thing1, thing1, thing1)
    print_three_things(thing2, thing2, thing2)
    print_three_things(thing3, thing3, thing3)
```
Example of Simple Flow of Control

```python
print_three_things_three_times('I','have','hair')

print_three_things('I','I','I') print_three_things('have','have','have') print_three_things('hair','hair','hair')

print('I') print('I') print('I') print('have') print('have') print('have') print('have') print('have') print('have')
```
Boolean Data Type and Logical Operators

- There are two objects of type *Boolean: True & False*
- Logical operators – operators which output Boolean values

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arguments</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>p == q</td>
<td>p and q are of any type</td>
<td>True iff p and q are equal</td>
</tr>
<tr>
<td>p != q</td>
<td>p and q are of any type</td>
<td>False iff p and q are equal</td>
</tr>
<tr>
<td>not p</td>
<td>p is of type boolean</td>
<td>True iff p is False</td>
</tr>
<tr>
<td>p and q</td>
<td>p and q are of type boolean</td>
<td>True iff both p and q are True</td>
</tr>
<tr>
<td>p or q</td>
<td>p and q are of type boolean</td>
<td>True if p is True or q is True or both are True</td>
</tr>
</tbody>
</table>

- *not* is a unary operator (occurs before its one argument)
- *or* is *inclusive or* not *exclusive or*
- *==* is a logical operator; *=* is the assignment operator
OR

• In English, the word *or* is ambiguous
  – *Are you a boy or a girl?*
    • *Both* would be an unusual answer
    • This kind of *or* is called *exclusive or*
  – *Do you own a hair dryer or a toaster oven?*
    • *Both* would be a normal answer
    • This kind of *or* is called *inclusive or*

• In python and most programming languages
  – *or* means *inclusive or* only

• However, we can define *exclusive or*
  – def xor (p, q):
    return((p or q) and (not (p and q)))
## More Boolean Operators (Math Only)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arguments</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &lt; y )</td>
<td>( x ) and ( y ) are integers/floats</td>
<td>True iff ( x ) is less than ( y )</td>
</tr>
<tr>
<td>( x \leq y )</td>
<td>( x ) and ( y ) are integers/floats</td>
<td>True iff ( x ) is less than or equal to ( y )</td>
</tr>
<tr>
<td>( x &gt; y )</td>
<td>( x ) and ( y ) are integers/floats</td>
<td>True iff ( x ) is greater than ( y )</td>
</tr>
<tr>
<td>( x \geq y )</td>
<td>( x ) and ( y ) are integers/floats</td>
<td>True iff ( x ) is greater or equal to ( y )</td>
</tr>
</tbody>
</table>
Boolean Data Type & Logical Operators 2

• Boolean values (and therefore logical operators) are used for conditional statements in programs
  – Different statements may activate depending on whether a variable has a *True* or *False* value
  – Or some cycle will repeat until a variable has a *True* or *False* value

• Logical Operators combine Boolean values together in various ways

• Truth Tables (from propositional logic) are useful for correctly interpreting combinations of Boolean values
Truth Table for combinations of $p$ & $q$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p == q$</th>
<th>$p != q$</th>
<th>$p$ and $q$</th>
<th>$p$ or $q$</th>
<th>not $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Order of Precedence: *not, and, or*  

- Use parentheses to avoid ambiguity when linking more than two expressions with `not`/`and`/`or`  
- Example ambiguity:  
  - `(Not True) or False or True == True`  
  - `Not (True or False or True) == False`  
  - `(True and False) or ((True or True) and False) == False`  
  - `(True and (False or True or (True and False))) == True`  
- Precedence: parentheses, `==`, `!=`, `not`, `and`, `or`  
- Using parentheses is easier for humans than relying on default precedence rules  
  - Ambiguity similar to situation in arithmetic
Conditionals: if and else

• These keywords divide blocks of statements
  – Based on the evaluation of Booleans as True or False
• For example, consider the following code

```python
def is_your_name_bruce(name):
    if (name == 'Bruce' or name == 'bruce'):
        print('Your name IS Bruce!')
        return(True)
    else:
        print("Well, I guess your name isn't Bruce, now is it?")
        return(False)
```
Syntax of *if* and *else*

- *if* is followed by:
  - A boolean expression, a colon and a block of text
    - The block of text is indented
    - We will call the boolean expression the *if statement*
  - The text block is executed if the boolean expression is true
    - We will call this the *then statement*
- Optionally, *else*: can introduce another text block
  - this executes if the boolean expression is false.
    - We will call this the *else statement*
Sample Application: Interactive Fiction

• The first adventure game was text based
  – It was written by W. Crowther in the 1975
  – Available for Windows:
  – Available for MAC:
    http://www.lobotomo.com/products/Adventure/index.html
  – Source code (Fortran): http://jerz.setonhill.edu/if/crowther/
  – Inspired by role playing game: Dungeons and Dragons (1974)

• Interactive version of *Goldilocks and the 3 Bears*
Intro to: Computers & Programming:
Booleans, Conditionals and Loops: Flow of Control in Python
V22.0002-001

Start

Starving and Exhausted, Goldilocks found a cottage in the middle of the forest. Nobody was home.

Should she enter it?

Yes

She finds some porridge and eats it. Then she breaks some furniture.

She wakes up surrounded by 3 angry bears.

No

After traveling for the rest of the day she found a cave and lived off roots and fungi until she was rescued.

She works for the bears as a servant for one month to pay for the damage.

The sheriff is called. Goldilocks goes to jail for trespassing and destruction of private property.

She should she apologize?

Yes

No

She’s tired. Should she rest for a few minutes in one of the beds?

Yes

No
Basic Idea for Program

• An interactive program
  – User answers a series of yes/no questions
  – Use the `input` function to get keyboard input
    ```python
    answer = input("Type 'yes' or 'no': ")
    ```
    • Sets the variable `answer` to the string input by the user

• Uses `if` and `else` to divide the `yes` and `no`
  choices in the flowchart.

• Most of the program involves printing different sections of the text.
goldilocks.py

- The program performs as in the flowchart
  - Some minor changes in the text

- The main function
  - Uses a series of nested instances of `if` and `else`
  - Calls `get_yes_or_no` to query if the user types `yes`
    - Allowances are made for imprecise responses
  - Calls `print_ending1` to print the most common ending
  - Some `if statements` are `(not get_yes_or_no())`
    - It is easier to read if the shorter path is listed first
      - Quick paths to the end are listed first
Minor Points about goldilocks.py

• The backslash \ is used to *logically* join multiple lines
  – Python is unusual in that syntax requires spacing, newlines and indents. So this is a way of offsetting the disadvantages.

• Some of the details of running this program are different for Python 3.x and Python 2.x
  • It won't run if the wrong version is executed
elif

• The following structure
  
  if x:
      else:
          if y:
              else:
                  if z: ....

• Can be abbreviated using elif (else if)
  
  if x:
      elif y:
          elif z: ....

• This can make the code easier to read
Binary Branching Decision Trees

• Complex decisions can be broken down into a series of yes/no questions, forming a binary branching tree.
• The graph on the following page suggests how the flow of control can proceed in such programs.
• Programs using decision trees can have a similar structure to the goldilocks program
• Only 4 out of the 15 questions in a binary decision tree are ever asked when the function is called.
  – The system asks $\log_2 N + 1$ out of $N$ questions
• Applications include expert systems (medical, automotive, etc.) and automatic teller machines
Binary Decision Trees

[Diagram of a binary decision tree with questions and leaf nodes.]
The `input` Function

- We went over this briefly, but it is important
- The function: `input(Prompt)`
  - Prints the string bound to Prompt
  - Returns a string entered by a user
- Example: `input('Name a Number: ')`
  - Notice the colon and space in the prompt
  - The string ends when the user hits [Enter]
  - It is a string not a number
Add Numbers Example

• `def add_numbers ():`
  ```python
  num1 = input('First Number: ')
  num2 = input('Second Number: ')
  return (int(num1) + int(num2))
  ```

• **Note:** `input` returns a string, not an integer
  – We used the function `int` for type conversion
Conditionals Can Be Used to Identify Errors in User Input

• The function `get_yes_or_no` in the Goldilocks program
  – If the answer is yes or Yes, return True
  – Else if the answer is no or No, return False
  – Else
    • Print “your answer is unclear, but we think you mean no”
    • Return False

• Other possibilities:
  – Use loops (coming up soon) to keep asking the user for more input until they provide well-formed input
  – Print “this is an error” and return “error”
Summary

• Flow of Control refers to the determination of when commands are executed. Factors include:
  – order of statements
  – order of the blocks containing statements
  – evaluation of boolean expressions in \textit{if} & \textit{elif} clauses
    • If the boolean evaluates to \textit{True}
      – The body of \textit{if}/\textit{elif} executes
    • Otherwise, the body of the following \textit{else} executes (if it exists)

• Flow of Control relies on boolean operators (==, !=, not, and, or), mathematical boolean operators (<,>,<=,>=) and other functions that return boolean values.
  – Parentheses recommended (or an understanding of precedence rules)

• The \textit{input} function provides a simple means of user interaction

• The \textit{decision tree} is a simple, but powerful algorithm for problem solving
Homework: Slide 1

• Due in 2 or 3 classes
• Read first ½ of Chapter 4
• Design a program that uses a decision tree
  – Write out your decision tree and include your plan as either a separate file or a set of comments
• Write a program that implements this decision tree
  – Interactive fiction or other game
  – A questionnaire that is geared towards solving a particular kind of problem (e.g., choosing a car)
  – An expert system for solving some problem that you are an expert on
  – A system for classifying objects
• Grading criteria provided on the next slide
Homework Slide 2: Grading Criteria

• Topic: interesting? A good fit?
• Planning
  – Did you implement what you planned?
  – Is it a good plan?
• The shape and size of the tree
  – How many questions are involved?
  – How deep is the tree?
• Does the program work?
• Did you do anything innovative?
• Is your code clear and well-written?