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: Decision Trees
• Loops: Next Topic
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

```
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')
```
Flow and Control and Boolean Values

- Sometimes a statement will only execute
  - `if` some expression evaluates to True or False
- `while` loops keep repeating a block of text until
  - A particular expression evaluates to True or False
- Other types of flow of control may also depend on True or False statements
- So first, we will describe the nature of Boolean Expressions, expressions which evaluate as True or False
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 short or tall?*
    - *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*
  - *Either/or* usually implies exclusive or, e.g., “You can have either a hair dryer or a toaster oven” implies that you can have only one of these

- In python and most programming languages
  - *or* means *inclusive or* only

- We can define *exclusive or* using operators *and* and *or* and *not*
  - `def xor(p, q):
            return((p or q) and (not (p and q)))`

- `p != q`
  - *Not equal* is equivalent to *Exclusive or* if *p* and *q* are Boolean expressions
Is a tomato a fruit *or* a vegetable?

- **Exclusive *or* Inclusive *or***?

  - **Botany**: inclusive or → *Both a fruit and a vegetable*
    - *Vegetable*: the edible parts of a plant; the name of the kingdom that plants belong to; members of the vegetable kingdom, …
    - *Fruit*: the mature ovary of a plant (which is often edible and thus also a vegetable)

  - **Culinary Arts**: exclusive or → *A fruit*
    - *Vegetable*: plant-based food that is not typically sweet nor part of dessert/breakfast
    - *Fruit*: similar to botany meaning, but typically sweet and eaten as part of dessert or breakfast
Math Operators returning Boolean values

<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>$x &lt;= y$</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>$x &gt;= y$</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

- Plug examples in table:
  - p = A tomato is a fruit
  - q = A tomato is a vegetable

<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 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>

- Notes:
  - == and != are used with all types of input
  - != with boolean input is **exclusive or**
  - == with boolean input is the **biconditional operator**: ↔
Order of Precedence: *not, and, or*

- Use parentheses to avoid ambiguity when linking more than two expressions with not/and/or

- Example ambiguity:
  - $(\text{Not True}) \text{ or } \text{False or True} == \text{True}$
  - Not $(\text{True or False or True}) == \text{False}$
  - $(\text{True and False}) \text{ or } (\text{(True or True) and False}) == \text{False}$
  - $(\text{True and (False or True or (True and False)))} == \text{True}$

- Precedence: parentheses, $==$, $!=$, not, and, or
  - Biconditional, Exclusive or, not, and, or $\rightarrow$ BENAO
  - Not the most intuitive acronym (although neither is PEMDOS)

- Using parentheses means not having to remember precedence rules
  - Ambiguity similar to situation in arithmetic
De Morgan's Laws

• Rules of Propositional Logic as Python statements, where X and Y are either True or False—make intuitive sense, but easy to forget

• \((\text{not } (X \text{ or } Y)) == ((\text{not } X) \text{ and } (\text{not } Y))\)

• \((\text{not } (X \text{ and } Y)) == ((\text{not } X) \text{ or } (\text{not } Y))\)

• Try all combinations of True and False
  – See sweet and salty example
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 *condition*
  - The text block is executed if the boolean expression is true
    - We will call this the *consequence*
- 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
  - A python version:  
    https://pypi.python.org/pypi/adventure/1.2
  - Inspired by role playing game: Dungeons and Dragons (1974)

- Interactive version of *Goldilocks and the 3 Bears*
  - Simpler code and simpler program, but the same basic idea.
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.

No: After traveling for the rest of the day she found a cave and lived off roots and fungi until she was rescued.

**Should she apologize?**

Yes: She wakes up surrounded by 3 angry bears.

No: She's tired. Should she rest for a few minutes in one of the beds?

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

No: She works for the bears as a servant for one month to pay for the damage.
Basic Idea for Program

• An interactive program
  – User answers a series of yes/no questions
    • (More possible answers in Adventure game)
  – Use the `input` function to get keyboard input
    `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
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 balanced binary decision tree are ever asked when the function is called.
  - In a balanced binary tree, 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
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 while 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”
Examples of Decision Tree Programs

- Goldilocks (previous slides)
  - Similar to: interactive fiction games, educational software, children's stories, adventure-type games
- Automatic Bank Teller Machines
- Expert Systems
- Automated Phone Systems
Common Error for logical operators

- Let's assume that a bag has only yellow and black M & Ms
- A simple program checks if the bag has your favorite color M & M
- If your favorite is 'green', the program should indicate that your M & M is not in the bag
- Note the difference between checking_your_m_and_m1 and checking_your_m_and_m2
Incorrect and Correct Functions

- def checking_your_m_and_m_incorrect(your_m_and_m):
  
  if favorite_m_and_m == 'yellow' or 'black':
    print('Yes your m & m is in the bag')
  else:
    print('No, we do not have your m & m')

- def checking_your_m_and_m_correct(your_m_and_m):
  
  if (favorite_m_and_m == 'yellow') or (favorite_m_and_m == 'black'):
    print('Yes your m & m is in the bag')
  else:
    print('No, we do not have your m & m')

- Difference
  - favorite_m_and_m == 'yellow' or 'black'
  - (favorite_m_and_m == 'yellow') or (favorite_m_and_m == 'black')
Interpretation of incorrect if statement

- if favorite_m_and_m == 'yellow' or 'black':
  print('Yes your m & m is in the bag')
- if (favorite_m_and_m == 'yellow') or 'black':
  print('Yes your m & m is in the bag')
- If non-booleans are part of logical statements
  - Most are interpreted as True, e.g., 'black' is equivalent to True
  - Certain special items are interpreted as False
    - " – empty string
    - NONE – output of function producing no output, e.g., `print`
    - The integer 0
    - Other smallest objects for types not covered in class yet, e.g., empty list
Example 2: A (toy) Expert System to Distinguish a Cold from the Flu

- Source:

- 1st Step: Sum up all the factors involved

- 2nd Step: Model them as a decision tree, an organized series of simple questions

- 3rd Step: Implement them as a Python program
Simplifications for Writing Program

• There will be the following possible results:
  – Cold
  – Swine Flu
  – Seasonal Flu
  – No detected illness

• The program will require at least 3 symptoms to make a diagnosis

• Symptoms divided into classes:
  1. Symptoms that absolutely favor Flu over Cold
  2. Symptoms that tend to favor Flu
     a. Symptoms that absolutely favor Swine Flu over Seasonal Flu
     b. Others
  3. Symptoms that tend to favor Cold

• Absolute symptoms predict an outcome (e.g. vomiting predicts Swine Flu)

• Other symptoms combine via voting scheme:
  – more cold symptoms → cold
  – more flu symptoms → flu
# Table from 2nd Page of Web-MD Article

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Cold</th>
<th>Flu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fever</td>
<td>Mild, more common in children</td>
<td>Usually Higher (100° to 102° F), lasts 3-4 days</td>
</tr>
<tr>
<td>Headache</td>
<td>Occasional</td>
<td>Common</td>
</tr>
<tr>
<td>Aches/Pains</td>
<td>Slight (implied not always)</td>
<td>Usual, often severe</td>
</tr>
<tr>
<td>Fatigue, weakness</td>
<td>Sometimes</td>
<td>Usual, can last 2 to 3 weeks</td>
</tr>
<tr>
<td>Extreme Exhaustion</td>
<td>Never</td>
<td>Usual, at beginning of illness</td>
</tr>
<tr>
<td>Stuffy Nose</td>
<td>Common</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Sneezing</td>
<td>Usual</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Sore Throat</td>
<td>Common</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Chest Discomfort, Cough</td>
<td>Mild to moderate; hacking cough (implied not always)</td>
<td>Common; can become severe</td>
</tr>
</tbody>
</table>
Two types of Flu: Seasonal and Swine Flu

Symptoms: sore throat, fever, headache, muscle ache, soreness, congestion, cough

Swine flu specific symptoms: vomiting and diarrhea

Duration: a few days to a few weeks

Symptoms can take a few days to a week to dissipate
Model informal info from WebMD article

• Use point system for words describing tendencies:
  – never = 0, occasional/sometimes/mild/slight = 1, common = 3, usual = 6, always = 10

• Severity of Symptoms can be treated the same way:
  – Nonexistent = 0, mild = 1, moderate = 5, extreme = 10

• Make assumptions to model human knowledge
  – Low fever: temperature $\geq$ 99 and temperature $\leq$ 100
  – High fever: temperature $>$ 100
  – Assume a child is a person $\leq$ 16 years old
Classes of Symptoms

• All symptoms:
  – fever, tiredness, headache, fatigue, other aches and pains, chest discomfort and coughing, stuffy/runny nose, sneezing, sore throat
  – These can be true/false or have a range of values

• Symptoms absolutely favoring Flu
  – High level of fatigue or high temperature

• Symptoms favoring cold
  – low fever in children, sneezing, sore throat, stuffy/runny nose

• Symptoms favoring flu
  – headache, other aches, medium level of fatigue, coughing, illness longer than 1 week

• Symptoms absolutely favoring Swine over Seasonal Flu
  – vomiting and diarrhea
Ways We Could Improve the Program

• Include information about the sequence of symptoms.
  – Colds often begin with sore throats, which go away after a few days and are followed by nasal symptoms

• Consult other articles

• Consult a doctor

• Test the accuracy of the program on real data (real instances where we know the diagnosis and symptoms).
  – Modify the program to better account for the data
  – Test the program on new data
User Input Functions

• **is** _yes_or_no_
  – One argument parameter: the question to be asked
  – Uses `input` function to ask the question and retrieve answer
  – Converts yes or no answer into True or False
  – Assumes unexpected answers are equivalent to 'No'
    • Alternative: Give user an error and exit the program
    • Alternative: Give user an error and ask for Yes or No again

• **number_from_zero_to_ten**
  – Makes sure that an integer from 0 to 10 is used
  – Rounds to the nearest integer
  – Error if the user entered a non-number
    • Error handling discussed later in term
More Implementation Details

- Counters: *symptoms, flu_symptoms, cold_symptoms*
  - Incremented by 1 when we identify a new symptom in a category
    - \( \text{counter} = 1 + \text{counter} \)
- Conditions contain boolean operators \((>, <, \geq, \leq, \==, \!=)\)
  - Example: \( \text{if symptoms} \geq 3: \)
- \text{check_for_swine_flu}
  - Exactly 2 symptoms implying Swine Flu (assuming flu)
  - One simple yes/no question covers this
  - Function used multiple times and easy to modify should we identify additional swine flu symptoms
Our System vs. Real Medical Expert Systems

• We need to include a warning in this program that the diagnosis should not be taken seriously
• Before releasing a real expert system, we would test it extensively and modify it so it performs accurately.  
  – There was no quality control for this program
• Doctors are consulted for real systems – systems are not based on web articles written for non-doctors
• One expert system designed by a doctor is available online at: http://easydiagnosis.com/
  – Dr. Schueler, who designed this system, also warns that this program should not take the place of a real doctor
Expert Systems

• These can be represented by decision trees
  – They attempt to model human reasoning based on the order in which a human being would ask questions.

• Of course, there are other models for automatically making the same sorts of decisions
  – For example, predictions can be based on statistical correlations

• They are used in many fields: medicine, fixing machinery, how to choose a wine, picking an airplane flight, etc.

• Information on expert systems: http://edutechwiki.unige.ch/en/Expert_system
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 if & elif clauses
    - If the boolean evaluates to True
      - The body of if/elif executes
    - Otherwise, the body of following elif or else clauses may execute
- Flow of Control relies on boolean operators (==, !, not, and, or), mathematical boolean operators (<, >, <=, >=) and other functions that return boolean values.
  - Parentheses recommended (easier than precedence rules)
- The input function provides a simple means of user interaction
- The decision tree is a simple, but powerful algorithm for problem solving: interactive fiction and expert systems are 2 common applications of the decision tree
Homework

• Go to HW part of website
  – 004: http://cs.nyu.edu/courses/spring17/CSCI-UA.0002-004/index.html#HomeworkSchedule
  – 009: http://cs.nyu.edu/courses/spring17/CSCI-UA.0002-009/index.html#HomeworkSchedule

• See HW due February 13