CMPT 120

Topic: Conditional Statements – Part 2
Last Lecture

• Introduce Python Statements #5
  -> Conditional Statements
    1. Syntax #1
    2. Syntax #2
    3. Syntax #3
Learning outcomes

At the end of this course, a student is expected to:

- Describe and apply fundamental concepts and terminology of Python:
  - Conditions and execution flow
- Create (design) small to medium size programs using Python:
  - Use the core features of Python to design programs to solve problems: conditionals
- etc...
Today’s Menu

- Continue exploring Conditional Statements
  - Syntax #4 -> Chained if
- Introduction to yet another data type -> bool
- How to construct conditions
Let’s practice:

• **Step 1 - Problem Statement**
  • Write a Python program that determines the day of the week, given a number representing a week day, where 0 represents Monday, … , 6 represents Sunday
Solution #1 – using nested if

```python
print("Consider the idea that we can represent a weekday using a number between 0 and 6.")
print("Enter a number between 0 and 6 and I shall tell you to which weekday that number corresponds.")
weekday = int(input("Your number is: "))
if weekday == 0:
    print("That's Monday!")
else:
    if weekday == 1:
        print("That's Tuesday!")
    else:
        if weekday == 2:
            print("That's Wednesday!")
        else:
            if weekday == 3:
                print("That's Thursday!")
            else:
                if weekday == 4:
                    print("That's Friday!")
                else:
                    if weekday == 5:
                        print("That's Saturday!")
                    else:
                        if weekday == 6:
                            print("That's Sunday!")
                        else:
                            print("Your number was not between 0 and 6 :(!")
```
Step 5 - Testing

• How many distinct test cases would we need in order to completely test Solution #1?
Review: How nested if works

- Conditional Statement – nested if statement
- Syntax #3

```python
if weekday == 0:
    print("That's Monday!")
else:
    if weekday == 1:
        print("That's Tuesday!")
    else:
        if weekday == 2:
            print("That's Saturday!")
        else:
            if weekday == 3:
                print("That's Sunday!")
            else:
                print("Your number was not between 0 and 3 :( !")
```

"Green" if else is nested in the "blue" if else

"Red" if else is nested in the "green" if else

"Black" if else is nested in the "red" if else
Solution #2 – using chained if

```python
print("Consider the idea that we can represent a weekday using a number between 0 and 6.")
print("Enter a number between 0 and 6 and I shall tell you to which weekday that number corresponds.")
weekday = int(input("Your number is: "))
if weekday == 0:
    print("That's Monday!")
elif weekday == 1:
    print("That's Tuesday!")
elif weekday == 2:
    print("That's Wednesday!")
elif weekday == 3:
    print("That's Thursday!")
elif weekday == 4:
    print("That's Friday!")
elif weekday == 5:
    print("That's Saturday!")
elif weekday == 6:
    print("That's Sunday!")
else:
    print("Your number was not between 0 and 6 :(")
```
Conditional Statement – Syntax #4

chained if statement

```
if <condition 1>
    <some statement(s)>
else if <condition 2>
    <some statement(s)>
else:
    <some statement(s)>
```
How chained if works

• Conditional Statement – chained if statement
• Syntax #4

```python
if weekday == 0:
    print("That's Monday!")
elif weekday == 1:
    print("That's Tuesday!")
elif weekday == 2:
    print("That's Saturday!")
elif weekday == 3:
    print("That's Sunday!")
else:
    print("Your number was not between 0 and 3 :( ")
```

“Green” if `elif` is chained to the “blue” if `elif`

“Red” if `elif` is chained to the “green” if `elif`

“Black” if `else` is chained to the “red” if `elif"
Questions (using Slide 10)

Which "statement(s)" will execute if...

1) Condition 1 is True :
2) Condition 2 is True :
3) Conditions 1 & 2 are False :
4) Conditions 1 & 2 are True :
5) Condition 1 is False & Condition 2 is True :
Conclusion
Back to our Guessing game

• We can use a chained conditional statement

• Solution: GuessingGame_2_C.py
Improving our solution

• Let’s improve our guessing game even more by making sure the user enters a number, not a string

• Solution: GuessingGame_3.py

• Step 5 – Testing
  • How would we test this program?
Improving our solution

- Can you think of other improvements we can make to our guessing game?
Conditions seen so far

• Example from our Login programs:
  
  userPassword == thePassword

• Example from our GuessignGame program:
  
  number == userGuess
  
  >
  <
  >=
  <=
Condition

• A condition is a **Boolean expression**

**Boolean expression:**
• An expression that produces a result that has a value “true” or “false”
• These values are of data type `bool`
  (`bool` for Boolean)
• In Python, “true” is `True` and “false” is `False`

We have numbers in numerical expressions (e.g., 3, 12.87, 6) and they produce numerical results. Similarly, we have Boolean values in Boolean expressions and they produce Boolean results.
How to construct a condition?

**SYNTAX:** `<operand> <operator> <operand>`

Relational operators (or comparison operators)

- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to
- `==` equal to
- `!=` not equal to

Result of conditional expression: **True** or **False**

Relational operators (<, >, <=, ==, ...) may connect expressions evaluating to different types of values
Let’s practice!

What is the result of the following ...

• Python Code Fragment 1
  ageMike = 25
  agePaul = 30
  print("ageMike != agePaul -> ",
        ageMike != agePaul)

• Python Code Fragment 2
  answer = "a"
  print("answer <= 'C' -> ", answer <= 'C')

How can we compare letters?
# ASCII Code and Unicode

<table>
<thead>
<tr>
<th>Dec</th>
<th>Char</th>
<th>Dec</th>
<th>Char</th>
<th>Dec</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>[space]</td>
<td>64</td>
<td>@</td>
<td>96</td>
<td>`</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>68</td>
<td>D</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>69</td>
<td>E</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>70</td>
<td>F</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>39</td>
<td>'</td>
<td>71</td>
<td>G</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>72</td>
<td>H</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>73</td>
<td>I</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td>74</td>
<td>J</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td>75</td>
<td>K</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td>76</td>
<td>L</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>77</td>
<td>M</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td>78</td>
<td>N</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>79</td>
<td>O</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>80</td>
<td>P</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>82</td>
<td>R</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td>83</td>
<td>S</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>84</td>
<td>T</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>85</td>
<td>U</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td>86</td>
<td>V</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
<td>87</td>
<td>W</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>88</td>
<td>X</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>57</td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>58</td>
<td>:</td>
<td>90</td>
<td>Z</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>59</td>
<td>;</td>
<td>91</td>
<td>[</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>60</td>
<td>&lt;=</td>
<td>92</td>
<td>\</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>=&gt;</td>
<td>93</td>
<td>]</td>
<td>125</td>
<td>}</td>
</tr>
<tr>
<td>62</td>
<td>?</td>
<td>94</td>
<td>^</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>63</td>
<td>?</td>
<td>95</td>
<td>_</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to construct a condition?

**SYNTAX:** .<method>(...) <function>(...) operator

Example:
<string>.isdigit() <string>.isalpha() etc...

Example:
all(...) etc...

Example: in containment test operator

Result of conditional expression: **True** or **False**

We have built-in functions returning numerical values:
- `len("hello")` returns 5
- `int("5")` returns 5

We have methods returning numerical values:
- "hello".find("lo") returns 3

Similarly, we have built-in functions returning Boolean values:
- `all([1<2,2<4,5==5])` returns True

We have methods returning Boolean values:
- "123456".isdigit() returns True
- "123456".isalpha() returns False
Variables and Boolean expressions

- A variable can contain a value of `int`, `float`, `str` or `list` type.
- It can also contain a value of `bool` type:
  - Example:
    ```python
    loggedIn = True
    ```
  - Example:
    ```python
    age = 25
    drinkingAge = age > 18
    ```
What if we have more than 1 conditions?

- In our Guessing Game
  - If the user enters a guess (a number) between 1 and 10 -> valid data
  - If the user enters a guess < 1 or > 10 -> invalid data

- Guardian code (input validation code):

  ```python
  if userGuess < 1 or userGuess > 10:
      print("Your guess should be a number between 1 and 10...")
      # then terminate program
  else:
      ...
  ```

This is called a **compound condition**: a condition that is composed of > 1 condition.
What if we have more than 1 conditions?

- We could also create the following guardian code (input validation code):

```python
if userGuess >= 1 and userGuess <= 10:
    # Did the user guess correctly?
    # Display appropriate message
    if number == userGuess:
        print("Wow! You got it!")
    else:
        print("Sorry! You missed! The number was \%d." %number)
else:
    ...
```
How to construct a logical expression (compound condition)?

- Conditions (Boolean expressions, i.e., expressions producing Boolean values), can be connected via the logical operators
  - **and**, **or**, **not**
  - creating logical expressions

- Such logical expressions are called **compound conditions**
How to construct a logical expression (compound condition)?

SYNTAX for:
- **AND** & **OR**: `<operand> <logical_operator> <operand>`
- **NOT**: not `<operand>`

**Logical operators**
- **Operand**
- **AND**
- **OR**
- **NOT**
- **Operand**

Result of compound conditional expression: True or False

**Boolean operators** **AND**, **OR** must connect two Boolean expressions
**Boolean operator** **NOT** must be applied to one Boolean expression
How to evaluate a logical expression (compound condition)?

- Boolean truth table:

  **and truth table**
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A and B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

  **or truth table**
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A or B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

  **not truth table**
  
<table>
<thead>
<tr>
<th>A</th>
<th>not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

# Revisiting order of evaluation:

<table>
<thead>
<tr>
<th>Highest precedence</th>
<th>Parentheses</th>
<th>Exponentiation</th>
<th>Multiplication, division, remainder</th>
<th>Addition and subtraction</th>
<th>Relational operators</th>
<th>Logical operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>(expressions...)</td>
<td>P</td>
<td>E</td>
<td>MDR</td>
<td>AS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x[index],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x[index : index],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x(arguments...)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*, /, //, %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+, -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, !=, ==</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lowest precedence
Let’s practice evaluating compound conditions
Summary

• Conditional Statements
  • Syntax #4 -> Chained if
• Construct conditions
• Boolean data type
• Compound conditions
Next lecture

• Iterative statements