Conditionals

CS 1111
Introduction to Programming
Spring 2019

[The Coder’s Apprentice, §6-6.2]
Based in part on “Agnostic Programming: Learning to Design and Test Basic Programming Algorithms” by Kinga Dobolyi, Kindle]
What is a Decision Statement?

- A statement that evaluates some conditions to true or false.

```python
if condition:
    statement
    statement ...
```

- A condition (or expression)
  - Must always evaluate to true or false, i.e., “Boolean expression”

1. age = get the age from the user
2. if age > 20:
   True:
   ```python
   result = "Congrats! You can now rent the Two Door Speck!"
   ```
   False:
   ```python
   result = "Enjoy your bicycle, uphill both ways in the snow."
   ```
3. return result

age 19
Calculations that Evaluate to Boolean Values

- `< ≤ > ≥` all evaluate to true or false
  
  \[
  3 < 2 \text{ is False}
  \]

- `==, !=` also evaluate to True or False
  
  \[
  3 == 3 \text{ is True} \\
  3 == 4 \text{ is False} \\
  \text{“jello”} \neq \text{“blue”} \text{ is True} \\
  5 == 5.0 \text{ is True} \\
  \text{type}(5) == \text{type}(5.0) \text{ is False} \\
  \text{‘5’} == \text{‘5.0’} \text{ is False} \\
  \text{type(‘5’)} == \text{type(‘5.0’)} \text{ is True}
  \]
Decision Structure

Simple structure

if condition:
  statements1
else:
  statements2

Dual Structure

if condition:
  statements1
else:
  statements2

Indent the block of statements (sometimes called “body”)
# Two Types of Decisions

<table>
<thead>
<tr>
<th>Sequence Decision</th>
<th>Nested Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>if condition:</td>
<td>if condition:</td>
</tr>
<tr>
<td>block of statements</td>
<td>block of statements</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>if condition:</td>
<td>elif:</td>
</tr>
<tr>
<td>block of statements</td>
<td>block of statements</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>if condition:</td>
<td>elif:</td>
</tr>
<tr>
<td>block of statements</td>
<td>block of statements</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>else:</td>
<td>else:</td>
</tr>
<tr>
<td>block of statements</td>
<td>block of statements</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Nested decisions remember the results of decisions made before them (in the same nesting)
- Independent decisions do not
if Statements in Python

- if is a keyword, the if statement must end in a colon
- What belongs to a particular if statement is indented

```python
if operation is addition:
    True: result = number1 + number2
    False: do nothing
if operation is subtraction:
    True: result = number1 - number2
    False: do nothing
if operation is multiplication:
    True: result = number1 * number2
    False: do nothing
if operation is division:
    True: result = number1 / number2
    False: x = 3
```
**elif Statements in Python**

- **elif** is a keyword; it stands for else if
- **elif** is attached to an if or elif before it, and indicates this elif is nested
- (you cannot have a standalone elif)
if versus elif

- The one on the left returns 6
- if–elif statements are nested, linked, and mutually exclusive.

- The one on the right returns 4
- The plain if statements are not mutually exclusive, don’t know about each other, and thus all if statements get executed.
**else statements**

1. `number1` = get the first number from the user
2. `number2` = get the second number from the user
3. if `((number1 - number2) is 1) or ((number1 - number2) is -1):
   True: `result = "consecutive"
   False: `result = "not consecutive"
4. `return result`

```python
num1 = input("Enter number1: ")
num2 = input("Enter number2: ")
if ((num1 - num2) is 1) or ((num1 - num2) is -1):
    result = "consecutive"
else:
    result = "not consecutive"
return result
```

- **else** is a keyword, linked to an **if** or **elif**, and get executed if the **if/elif** above it is false
else statements (2)

```python
if operation == "addition":  
    result = number1 + number2
elif operation == "subtraction":  
    result = number1 - number2
elif operation == "multiplication":  
    result = number1 * number2
elif operation == "division":  
    result = number1 / number2
else:
    result = "operation undefined"
```

- else only gets executed if none of the if or elif before it are true
Indentation Matters

```python
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
            result += " finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))
print(template(1, 2))
print(template(2, 1))
```

This is another type of “nesting”, and is usually referred to as “nested if-else statements”
```python
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
    else:
        result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))  # "num1 is 1 num2 <= 3 finished num1"
print(template(1, 2))
print(template(2, 1))
```
Indentation Matters

def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
    result += "finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result

print(template(0, 1))
print(template(1, 3))
print(template(1, 2))
print(template(2, 1))
Indentation Matters

```python
def template(num1, num2):
    result = ""
    → if num1 == 0:
        result = "num1 or 0"
    → elif num1 == 1:
        result = "num1 is 1"
        → if num2 > 3:
            result += "num2 > 3"
        → elif num2 > 4:
            result += "THIS WILL NEVER RUN"
    → else:
        → result += "num2 <= 3"
        → result += " finished num1"
    → else:
        → result += "num1 is not 0 or 1"
    → return result

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))  # "num1 is 1 num2 <= 3 finished num1"
print(template(1, 2))  # "num1 is 1 num2 <= 3 finished num1"
→ print(template(2, 1))  # "num1 is not 0 or 1"
```

Indentation groups things

if, elif, and else are mutually exclusive
Unreachable statements

```python
def template(num1, num2):
    result = ""
    → if num1 == 0:
        result = "num1 or 0"
    → elif num1 == 1:
        result = "num1 is 1"
        → if num2 > 3:
            result += "num2 > 3"
    → elif num2 > 4:
        result += "THIS WILL NEVER RUN"
    else:
        result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result
```

```
num1 1
num2 5
result "num1 is 1 num2 > 3 finished num1"
print(template(1, 5))
```
• Assignment statement \( x = "\text{CS1111}" \)
• Boolean expression \( x \equiv "\text{CS1111}" \)
Boolean Types

```python
def boolean_example():
    value1 = (1 == 1)
    value2 = True
    value3 = False
    return value1 and value2 and not value3
```

- **True** and **False** are both keywords and types in Python
  - Capitalization !!

- **not** is a keyword that negates a Boolean value

- The code above returns **True**
Boolean Values and Calculations

- A boolean value must evaluate to true or false
- Two boolean values can be compared with \textbf{and} or \textbf{or}
- Use parentheses if you want to combine \textbf{and} or \textbf{or} to disambiguate; e.g., \((x \text{ and } y) \text{ or } z\) or \(x \text{ and } (y \text{ or } z)\)

\[
x = F, \ y = T, \ z = T
\]

\[
\begin{array}{cccc}
 F & F & F \\
 T & F & T \\
 F & T & F \\
 T & T & T \\
\end{array}
\]