Chapter 3: Branching and Program Flow

CSCI-UA 0002 – Introduction to Computer Programming

Mr. Joel Kemp
Agenda

• Logical Operators
• Boolean Operators
• if statements
• if/else and elif statements
Logical Operators

• Statements with logical operators evaluate to a boolean value (True or False).
  – Such statements are known as conditions or predicates.
  – Statements that test for:
    • Equality $A == B$  # is A equal to B?
    • Not Equal $A != B$
    • Greater Than $A > B$
    • Greater or Equal $A >= B$
    • Less Than $A < B$
    • Less or Equal $A <= B$

• Example:
  
  $A, B = 4, 5$
  
  `is_equal = A == B`  # Remember double equal signs!
  `is_greater = A > B`
Boolean Expressions

• We can combine relational statements into complicated expressions.
  – Boolean operators:
    • and
    • or
    • not

• Format:

  
  \((Boolean_{Exp\_1}) \text{ and } (Boolean_{Exp\_2})\)

  
  \((Boolean_{Exp\_1}) \text{ or } (Boolean_{Exp\_2})\)

  
  \(\text{not}(Boolean_{Exp\_1})\)
and

• What is it?
  – A boolean operation that evaluates to true only when all parts of the expression are true.

• Format:
  \((\text{Boolean}_\text{Exp}_1) \text{ and } (\text{Boolean}_\text{Exp}_2)\)

• Example:
  \[
  \text{time\_free = int(input(\text{"Num hours free: \")})}
  \]
  \[
  \text{is\_bum = time\_free >= 9 and time\_free <= 12}
  \]

• 😎 tip: Python lets you rewrite this as:
  \[
  \text{is\_bum = 9 <= time\_free <= 12}
  \]
• What is it?
  
  – A boolean operation that evaluates to true when **any** part of the expression is true!

• Format:

  \[(\text{Boolean}_\text{Exp}_1) \text{ or } (\text{Boolean}_\text{Exp}_2)\]

• Example:

  ```python
  hours_home = int(input("Num Hours Home: "))
  status = input("Occupation: ")

  is_hermit = hours_home >= 10 or status == "Hacker"
  ```
• What is it?
  – A boolean expression that evaluates to true only when the expression is false.
  – It’s the opposite of the expression’s truth value.

• How do we use it?
  
  not(Boolean_Exp)

• Examples:
  
  status = input("Education status: ")
  is_richer_than_joel = not(status == "PhD Student")

• Equivalent Alternative:
  
  is_richer_than_joel = status != "PhD Student"
If/ Else Statements

• Simulates a cause-and-effect.
  – If a condition is true, then do something.
  – Otherwise, do something else.

• We also call it branching!
**if Statement**

• What is it?
  – A branching construct whereby a set of code is executed based on the truth value of a condition.

• Syntax:
  ```python
  if condition == True:
      # Do the code here!
  ```

• What does it mean?
  – If the boolean condition/expression is **true**, then we execute the instructions within the **body** of the `if` statement.
  – Otherwise, we skip all of the code.

• But wait, how can we determine the body of an `if` statement?
  – Colon and Indentation
Colon and Indentation

• The colon : tells the interpreter to expect a body of statements.
  – Note: technically, the language allows us to ignore the colon, but it increases readability and allows IDE’s to better recognize the start of a block.

• Indentation is essential in associating statements with a particular block.

```python
if x > 5:
    print("It’s the truth")
print("I promise")
```
– The first print is associated with the if statement while the second print is on its own!
if else statements

• What if we have 20 consecutive if statements in our program?
  – Each if statement gets visited regardless of whether or not its body gets executed...
    • If the first condition evaluated to true, then we would still check the conditions for the other 19...

• If we find a true condition, can’t we just skip the others?
  – if else statements to the rescue!
if...else continued!

• Easy example: `else` as a fallback

```python
if(x == y):
    print(x, "is equal to", y)
else:
    print(x, "is not equal to", y)
```

• More involved example: combining `else` and `if`

```python
if(x == y):
    print(x, "is equal to", y)
elif (x != y):
    print(x, "is not equal to", y)
```

• Note: `else` and `elif` statements are optional
  – Use them only when necessary.

Refer to:
usingIfElse.py
usingIfElse2.py
THE GUESSING GAME
The Game

• Design a text-based game that asks the user to guess a number!
  – The system then outputs whether the guess was right or wrong!

• We’ll use our basic knowledge of branching to achieve this gameplay.

• Let’s solve this in parts:
  1. Output “Please guess a number: “ and input the user’s integer guess.
  2. Check if the user’s guess is the secret number
     • If true, output “You Win!”
     • Otherwise, output “You Lose!”
The Code

1. Output “Please guess a number: “ and input the user’s integer guess.
   
   userGuess = int(input("Please guess a number: "))

2. Check if the user’s guess is the secret number.

   secretNumber = 5  # Our little secret!

   • If true, output “You Win!”

   if(userGuess == secretNumber):
       print("You Win!")

   • Otherwise, output “You Lose!”

   else:
       print("You Lose!")
Finished Code

```python
secret = 5

userGuess = input("Please guess a number: ")
userGuess = int(userGuess)

if(userGuess == secret):
    print("You Win!")
else:
    print("You Lose!")
```

Let’s run it: guessingv1.py
A Smarter Program

• That was fun and impressive!
  – But, the program only told us whether the guess was right or wrong.

• Let’s make the program give some help to the user!
  – i.e. output whether the guess was **higher** or **lower** than the secret number!

<table>
<thead>
<tr>
<th>Number Entered</th>
<th>Message to Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Too Low</td>
</tr>
<tr>
<td>5</td>
<td>You Win!</td>
</tr>
<tr>
<td>7</td>
<td>Too High</td>
</tr>
<tr>
<td>8</td>
<td>Too High</td>
</tr>
</tbody>
</table>
Exercise

• Make our program smarter!

secret = 5

userGuess = input("Please guess a number: ")
userGuess = int(userGuess)

if(userGuess == secret):
    print("You Win!")
else:
    print("You Lose!")

<table>
<thead>
<tr>
<th>Number Entered</th>
<th>Message to Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Too Low</td>
</tr>
<tr>
<td>5</td>
<td>You Win!</td>
</tr>
<tr>
<td>7</td>
<td>Too High</td>
</tr>
<tr>
<td>8</td>
<td>Too High</td>
</tr>
</tbody>
</table>

Work on: guessingExercise.py
Revised Code

```python
secret = 5

userGuess = input("Please guess a number: ")
userGuess = int(userGuess)

if(userGuess == secret):
    print("You Win!")
elif(userGuess > secret):
    print("Too High!")
else:
    print("Too Low!")
```

Let’s run it:
guessingv2.py
The Problem

• Why do we have to keep rerunning the program every time we want to play?!
  – It would be great if the player could just keep guessing until they win!

• How can we get the computer to do something repeatedly?

Looping!