DECISION STRUCTURES & BOOLEAN LOGIC

CSCI-UA.002
Housekeeping

- Going above and beyond on projects?
  - Sure! Just prove you know what you’re talking about
  - Don’t make the code less efficient to demonstrate something more complex

- Late assignments?
  - 7-day grace period will start now
  - No late marks for first week
Sequence Structures

- What we have been programming so far is known as a “sequence structure”
- Sequence structures are sets of statements that execute in the order in which they appear
- Unfortunately not all programs can be written this way, as there are certain times when we need to deviate from a linear structure and adapt our program based on information provided.
Zombies are everywhere

Is this guy a zombie? (Yes/No)

Yes: Can you escape? (Yes/No)
  Yes: Hide in a small confined space with a group of people from different backgrounds
  No: Death

No: Yes he is
Example: Calculating Overtime Pay

- If a worker works more than 40 hours in a week he or she is entitled to overtime pay.
- Overtime pay is calculated at the rate of 1.5 times the worker’s hourly rate.
- This additional rate is only applied to hours worked above the 40 hour limit.
Example: Calculating Overtime Pay

- Input: Hourly rate of pay
- Input: Number of hours worked in 1 week

- Process: If the hours worked is less than 40, simply multiply hourly rate by hours worked
- Process: If the hours worked is greater than 40:
  - Multiply hourly rate by hours worked for 40 hours.
  - Subtract 40 from the total hours to obtain the overtime hours
  - Multiply overtime hours by 1.5 times the rate of pay
  - Add overtime pay to base pay

- Output: Total Pay
Example: Calculating Overtime Pay

- Our current Python toolset doesn’t give us the ability to deviate from a linear sequence structure.
The Selection Statement

- Allows your program to “ask a question” and respond accordingly.
- Simplest form – perform an action only if a certain condition exists
- If the condition is not met, then the action is not performed
The Selection Statement

- Start
  - Cold Outside?
    - True: Put on a coat
    - False: Finish
The Selection Statement

Start

- Cold Outside? True
  - Put on a coat
  - Wear a hat
  - Put on gloves

- Cold Outside? False
  - Finish
Selection Statements in Python

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

- **condition to be tested**
- **colon denotes end of condition**
- **statements to execute if condition is true**
AND BY TRUE, I MEAN FALSE.
Writing a condition

- The trick to writing a selection statement is in constructing a condition that matches the question you are trying to ask the computer.
- Think of conditions as “yes or no” questions. They can only be answered by one of two options – “True” or “False”.

```python
if condition:
```
Boolean Expressions

- Named after George Boole, a 19th century English philosopher and mathematician
- Boole developed a system of mathematics that allows us to work with the abstract concepts of “true” and “false”
- Boole is considered one of the founders of modern computer science, as his work underpins the way in which modern computers process binary data
Writing a Boolean Expression

- Boolean expressions can be used as the condition in an “if” statement.
- They are generally formed using “relational operators” which allow you to test to see whether a specific relationship exists between two (or more) values.
Relational Operators

\[ a > b \quad \# \text{ is a greater than } b \quad ? \]

\[ a < b \quad \# \text{ is a less than } b \quad ? \]

\[ a == b \quad \# \text{ is a equal to } b \quad ? \]

\[ a <= b \quad \# \text{ is a less than OR} \quad \# \text{ equal to } b \quad ? \]

\[ a >= b \quad \# \text{ is a greater than OR} \quad \# \text{ equal to } b \quad ? \]
Writing a Boolean Expression

- ALL Boolean expressions boil down to “True” or “False”
- Programmers often say that the expression “evaluates” to “True” or “False”
Writing a Boolean Expression

pen = 10
sword = 7

if pen > sword:  # pen > sword
    print ('the pen is mightier than the sword!')  # 10 > 7  # True
Let’s Evaluate!

# given these variables
a = 99
b = 7
c = -5
d = 92

# evaluate these expressions
a > b
b < c
b >= c
c <= d
a == b + d
d <= a + c
c != b
Boolean Operator Tips

- Don’t confuse “==“ with “=“
  - “=“ is used for assigning values to variables
  - “==“ is used for testing to see if two values are identical
- Use “!=“ if you want to test if two values are different
- The “<=" and “>=“ operators test for more than one relationship
  - “<=" tests to see if a value is less than OR equal to another
  - “>=“ tests to see if a value is greater than OR equal to another
Guppies are hardy fish, but they can’t live in all water temperatures.

The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.

Write a program that asks the user for a temperature. Then display one of two messages based on the information provided:

- You’re going to freeze your guppy!
- You’re going to boil your guppy!
The IF – ELSE structure
The selection statements we have been writing so far have only allowed us to create a single alternate branch of execution.

There are many times when we need to create multiple branches of execution based on the value of a Boolean expression.
The IF-ELSE structure

- The IF-ELSE structure allows you to perform one set of statements if a condition is true, and another if it is false
The IF-ELSE structure

Start

Input Salary

Salary > 50,000?

True

Qualify user for a loan

False

print "Sorry, you didn't qualify"

Finish
The IF-ELSE structure

```python
if temperature < 32:
    print ("it’s freezing outside!")
else:
    print ("it’s not so bad outside ...")
```
Nested Decision Structures
Sometimes you need to ask “follow up” questions after you’ve evaluated the value of a Boolean expression.

Python allows you to “nest” decision structures inside one another, allowing you to evaluate additional conditions.
Guess the Number using Nested Decision Structures

Start

Input a number

True

Number == 5 ?
False

Number < 5 ?

True

print "too low!"

False

print "too high!"

Finish
Nested Decision Structures

- Indentation is key – Python will use the indentation level of a structure to determine its relationship to any previous statements.
Programming Challenge: Freezing / Boiling / OK Guppies

- Guppies are hardy fish, but they can’t live in all water temperatures.
- The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.
- Write a program that asks the user for a temperature. Then display one of three messages based on the information provided:
  - You’re going to freeze your guppy!
  - You’re going to boil your guppy!
  - Your guppy is going to be fine!
IF-ELIF-ELSE Structure
Testing a series of conditions

- Testing a series of conditions using an IF-ELSE structure can result in a large amount of indentations

- Sometimes this can cause your code to become difficult to read

- Example: Grade determination program
  - Input: ask the user for a numeric grade (i.e. 95)
  - Process: convert the grade to its letter format (A through F)
  - Output: print the letter grade
Grade Determination Program

```python
# Grade Determination Program

g = float(input('grade '))

if (g > 90):
    print ('A')
else:
    if (g > 80):
        print ('B')
    else:
        if (g > 70):
            print ('C')
        else:
            if (g > 60):
                print ('D')
            else:
                print ('F')
```
IF-ELIF-ELSE

- You can simplify complex IF statements by using the ELIF structure
- ELIF is an optional structure that can be placed between your IF and ELSE statements
- It allows you to evaluate additional conditions at the same level as the original IF statement
if g > 90:
    print ('A')
elif g > 80:
    print ('B')
elif g > 70:
    print ('C')
elif g > 60:
    print ('D')
else:
    print ('F')
Some notes about using ELIFs:

- Conditions are tested in the order in which they are written. Once a condition evaluates to True all future conditions are skipped.

- An ELSE statement at the end of a decision structure is considered the “catch all” statement – if all conditions above end up failing then the statements inside the ELSE block will execute.

- However, using an ELSE statement at the end of your decision structure is optional.

- There is no logical need for an IF-ELIF-ELSE statement. You can always write a program without it by using a standard IF-ELSE block. The advantage of an IF-ELIF-ELSE statement is that your code may end up being more readable / understandable.
String Comparison
String Comparison

- So far we have been writing Boolean expressions that evaluate based on numeric data
  
  - Example: `x > 5; y < 10; z == 100`

- We can also construct Boolean expressions that can test relationships between strings

- When we compare strings we are essentially reducing them to their zeros and ones and comparing them numerically
<table>
<thead>
<tr>
<th></th>
<th>NUL</th>
<th>16</th>
<th>DLE</th>
<th>32</th>
<th>SP</th>
<th>48</th>
<th>0</th>
<th>64</th>
<th>@</th>
<th>80</th>
<th>P</th>
<th>96</th>
<th>`</th>
<th>112</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOH</td>
<td>17</td>
<td>DC1</td>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>18</td>
<td>DC2</td>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>19</td>
<td>DC3</td>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>20</td>
<td>DC4</td>
<td>36</td>
<td>$</td>
<td>52</td>
<td>4</td>
<td>68</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
<td>d</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>21</td>
<td>NAK</td>
<td>37</td>
<td>%</td>
<td>53</td>
<td>5</td>
<td>69</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
<td>e</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>22</td>
<td>SYN</td>
<td>38</td>
<td>&amp;</td>
<td>54</td>
<td>6</td>
<td>70</td>
<td>F</td>
<td>86</td>
<td>V</td>
<td>102</td>
<td>f</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>23</td>
<td>ETB</td>
<td>39</td>
<td>'</td>
<td>55</td>
<td>7</td>
<td>71</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
<td>g</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>24</td>
<td>CAN</td>
<td>40</td>
<td>(</td>
<td>56</td>
<td>8</td>
<td>72</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
<td>h</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>25</td>
<td>EM</td>
<td>41</td>
<td>)</td>
<td>57</td>
<td>9</td>
<td>73</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
<td>i</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>26</td>
<td>SUB</td>
<td>42</td>
<td>*</td>
<td>58</td>
<td>:</td>
<td>74</td>
<td>J</td>
<td>90</td>
<td>Z</td>
<td>106</td>
<td>j</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>27</td>
<td>ESC</td>
<td>43</td>
<td>+</td>
<td>59</td>
<td>;</td>
<td>75</td>
<td>K</td>
<td>91</td>
<td>[</td>
<td>107</td>
<td>k</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>28</td>
<td>FS</td>
<td>44</td>
<td>,</td>
<td>60</td>
<td>&lt;</td>
<td>76</td>
<td>L</td>
<td>92</td>
<td>\</td>
<td>108</td>
<td>l</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>29</td>
<td>GS</td>
<td>45</td>
<td>-</td>
<td>61</td>
<td>=</td>
<td>77</td>
<td>M</td>
<td>93</td>
<td>]</td>
<td>109</td>
<td>m</td>
<td>125</td>
<td>}</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>30</td>
<td>RS</td>
<td>46</td>
<td>.</td>
<td>62</td>
<td>&gt;</td>
<td>78</td>
<td>N</td>
<td>94</td>
<td>^</td>
<td>110</td>
<td>n</td>
<td>126</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>31</td>
<td>US</td>
<td>47</td>
<td>/</td>
<td>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
<td>127</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Boolean Operators for Strings

'dog' > 'cat'  # is 'dog' greater than 'cat'?

'fish' < 'alligator'  # is 'fish' less than 'alligator'?

'elephant' == 'tiger'  # are 'elephant' and 'tiger' equivalent?

'bat' != 'honey badger'  # are these strings different?

'bat' > 'back'  # is 'bat' greater than 'back'?
Programming Challenge: Password Protection

- Write a program that asks the user for a password
- Check to see if the password that was submitted is equal to the string ‘secret’
- If it is, print out a “welcome” message
- Otherwise, tell them to try again
Basic string manipulation

- Python has a huge string manipulation library that allows you to interact with and modify strings. We are going to get more in depth with this package later in the semester.
- For now we will only be exploring two small functions in this package – lower() and upper()
- The lower() function converts the characters in a string to all lowercase, while the upper() function converts the characters in a string to all uppercase.
- These functions are not built into the Python library directly, but exist inside the “str” module – as such they must be referred to using “dot syntax”
Basic String Manipulation

- Example:

```python
string_lc = str.lower('Harry Potter')
string_uc = str.upper('Harry Potter')

print(string_lc)
print(string_uc)
```
Programming Challenge: Case insensitive password

- Rewrite your password protection program to be case insensitive (i.e. the password “Secret” will also let you into your program)
String Length

- You can ask Python to count the number of characters contained in a string using the `len()` function.
- `len()` returns an integer that represents the total length of a string.
- Example:

```python
myname = 'harry'
print (len(myname))  # 5
```
Logical Operators
Logical Operators

- All programming languages provide a set of “logical operators”
- These operators can be used to create complex Boolean expressions that evaluate more than one condition at the same time
Logical Operators

```python
x = 10
y = 5
a = 20
b = 25

if x > y and a < b:
    print ('yes!')
else:
    print ('no!')
```
Logical Operators

- Logical operators are used to combine Boolean expressions into a composite Boolean expression.
- There are three main logical operators that we use regularly in programming:
  - and
  - or
  - not
The “and” operator

- “and” can be used to combine two Boolean expressions.
- The resulting Boolean expression will evaluate to be True if the two Boolean expressions it is connecting both evaluate to be True.

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True and True</td>
<td>True</td>
</tr>
<tr>
<td>True and False</td>
<td>False</td>
</tr>
<tr>
<td>False and True</td>
<td>False</td>
</tr>
<tr>
<td>False and False</td>
<td>False</td>
</tr>
</tbody>
</table>
Let's evaluate!

a = 5
b = 10

print (a > b and a > 1)
print (a > 1 and b > a)
print (a == 5 and b < 100)
print (a > 1 and b < 1 and b > a)
print (a > 1 and b > 1 and b > a)
“and” Example

```python
salary = float(input('How much do you make? '))
years = float(input('How long have you been at your job? '))

if salary >= 50000 and years >= 2:
    print ('You qualify for a loan!')
else:
    print ('You do not qualify for a loan')
```
The “or” operator

- “or” can also be used to combine two Boolean expressions
- The resulting Boolean expression will evaluate to be True if EITHER of Boolean expressions it is connecting evaluates to be True

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True or True</td>
<td>True</td>
</tr>
<tr>
<td>True or False</td>
<td>True</td>
</tr>
<tr>
<td>False or True</td>
<td>True</td>
</tr>
<tr>
<td>False or False</td>
<td>False</td>
</tr>
</tbody>
</table>
Let’s evaluate!

```python
a = 5
b = 10
print (a > b or a > 1)
print (a > 1 or b > a)
print (a == 5 or b < 100)
print (a > 1 or b < 1 or b > a)
print (a > 1 or b > 1 or b > a)
```
"or" Example

temp = float(input('What is the temperature of your fish tank? '))

if temp < 72 or temp > 86:
    print("The temperature is too extreme!")
The “not” operator

- The “not” operator is a unary operator that reverses the logical value of its argument.
- This means that it will “flip” a True value into a False value, and vice versa.
username = input('username? ')

if not (username == 'Harry'):
    print("invalid input!")
else:
    print("Welcome, Harry!")
“not” vs. `!=`

- Partially readability, but mostly this:

<table>
<thead>
<tr>
<th>Order of Operations</th>
<th>Operator(s)</th>
<th>Operator Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done 1st</td>
<td>()</td>
<td>specify order of operations</td>
</tr>
<tr>
<td>2nd</td>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>3rd</td>
<td>*, /, //, %</td>
<td>multiplicative</td>
</tr>
<tr>
<td>4th</td>
<td>+, -</td>
<td>addition</td>
</tr>
<tr>
<td>5th</td>
<td>==, !=, &lt;=, &gt;=, &lt;, &gt;</td>
<td>comparison</td>
</tr>
<tr>
<td>6th</td>
<td>not</td>
<td>logical</td>
</tr>
<tr>
<td>7th</td>
<td>and</td>
<td>logical</td>
</tr>
<tr>
<td>Done last</td>
<td>or</td>
<td>logical</td>
</tr>
</tbody>
</table>
Let’s write some programs!
Programming Challenge: Number Guessing Game (part 1)

- Ask the user to guess a number between 1 and 10. Assume they will enter an Integer.
- Pick a number between 1 and 10 that is your “secret” number (for example, 5)
- If the user types in your secret number, tell them that they win!
- If the user types in a number less than or greater than your secret number, tell them that they’re either above or below the number and to try again
Programming Challenge: Calculating a bonus

- You’re the manager of a large, distributed sales force
- You want to create an easy to use tool that will allow your sales staff to do the following:
  - Input their monthly sales amount
  - Determine if they made their monthly quota of $10,000
  - If they made their quota, they are eligible for a bonus of $500
  - If they made their quota, they should receive a “Good Job!” message
  - At the end of the program you should print out how much their bonus will be ($0 or $500)
Programming Challenge: Calculating a bonus

Input monthly sales

Met quota?

Assign bonus of $500

Print "You made your quota!"
Extension

- All sales people should receive 1% commission on their sales
- If a sales person made over 50,000, they should receive 5% commission on their sales (instead of 1%) – this is in addition to their $500 bonus for making their quota
- Print out their total take-home amount (bonus + commission) at the end of the program
Selection Statements in the Wild!

- How are selection statements used in ATM machines?
- How many selection statements can you count from your last ATM transaction?
Programming Challenge: Calculating Overtime Pay

- If a worker works more than 40 hours in a week he or she is entitled to overtime pay.
- Overtime pay is calculated at the rate of 1.5 times the worker’s hourly rate.
- This additional rate is only applied to hours worked above the 40 hour limit.
Programming Challenge: Calculating Overtime Pay

- **Input**: Hourly rate of pay
- **Input**: Number of hours worked in 1 week

- **Process**: If the hours worked is less than 40, simply multiply hourly rate by hours worked
- **Process**: If the hours worked is greater than 40:
  - Multiply hourly rate by hours worked for 40 hours.
  - Subtract 40 from the total hours to obtain the overtime hours
  - Multiply overtime hours by 1.5 times the rate of pay
  - Add overtime pay to base pay

- **Output**: Total Pay
Programming Challenge: Alphabetize two strings

- Ask the user to type in two names
- Compare the names and print them out in alphabetical order
Programming Challenge: Comparing the size of two strings

- Ask the user to input two names
- Sort the names in size order and print them out to the user
Programming Challenge

- Re-write the “guess the number” game using a nested decision structure.
- If the user guesses the number they win. If they don’t you should tell them to guess higher or lower next time depending on their answer.
secretnumber = 5

usernumber = int(input('Guess a number '))

if usernumber == secretnumber:
    print("you guessed it!")
else:
    if usernumber < secretnumber:
        print("your number is too low")
    else:
        print("your number is too high")
Programming Challenge

- Write a program that asks the user to enter in a number greater than or equal to zero and less than or equal to 100. If they do not you should alert them and end the program.

- Next, determine the letter grade associated with the number. For example, and A is any grade between 90 and 100. Report the letter grade to the user.
Programming Challenge: Loan Qualification

- You’re working for a small bank that wants to write a program to allow its customers to pre-qualify themselves for a personal loan

- Rules for qualification are as follows:
  - Borrower must make more than $50,000 per year and be at his or her job for at least 2 years
  - The 2 year job requirement can be waived, however, for borrowers making more than $100,000 per year

- Write a program to ask the user for their yearly salary as well as the # of years they have been at their current company. Use the rules above to output the string ‘You qualify’ or ‘You do not qualify’
Guess the Number using Nested Decision Structures

Start

Input a number

- True: Number == 5
  - True: print "You guessed the number!"
  - False: Number < 5
    - True: print "too low!"
    - False: print "too high!"

Finish
Guess the Number using Nested Decision Structures

```python
secretnumber = 5

usernumber = int(input('Guess a number '))

if usernumber == secretnumber:
    print("you guessed it!")
else:
    if usernumber < secretnumber:
        print("your number is too low")
    else:
        print("your number is too high")
```
Programming Challenge: Username and Password

- Write a program that asks a user for a username and a password
- Check to see if BOTH the username and password are correct
- If so, provide a Welcome message to the user
- If not, provide a Login Failure message to the user