styleguide.py
A style guide for assignments for Intro to Programming section 7

Written by: David Gochfeld
Written with: Craig Kapp  (include this if you worked with anyone else)

Date: 6 Feb, 2017  (or 2017-02-06)

# use comments liberally to explain what you are doing, and especially
# whenever it is not clear or when your code relies on assumptions
# that are not explicit

# use variable names that help the reader understand what the variable holds,
# how it will be used, and, if it's not clear, what type it contains

# get input from user
input_str = input("Please enter a value: ")

# assume it is a valid integer (will throw error otherwise)
input_num = int(input_str)

# put blank lines between sections of code that do different things,
# like paragraphs in an essay
# also use comments to to delineate sections of code by describing what they do

# calculations:
# first, raise two to the power of input_num
output_num = 2 ** input_num

# output the result
print("2 **",input_num,"=",output_num)
Decision Structures &
Boolean Logic

CSCI-UA.002
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.
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

- In this program we begin by asking a question – “is it cold outside?”

- If the answer to this question is yes (aka “True”) then we can execute an alternate set of commands.

- Otherwise we can continue with the program as-is.
The Selection Statement

Start

Cold Outside?

True
- Put on a coat
- Wear a hat
- Put on gloves

False

Finish
The Selection Statement

Start

Get Temp.

Temp > hi_thresh?

YES

Turn on A/C

NO

Temp < lo_thresh?

YES

Turn on Heat

NO

Turn off whatever is on.
Selection Statements in Python

"if" keyword begins a selection statement

condition to be tested

if condition:

statement
statement
statement

colon denotes end of condition

statements to execute if condition is true

"block" of execution must be indented
A feature of procedural programming languages

A group of statements with a beginning and end

if (x > y):
    first statement in block
    another statement
    last statement in block

In Python a block is indented, and ends when the indentation goes back to the previous level

Python blocks follow specific statements, like if
Boolean Expressions
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.

- All selection statements must have a condition to “test”.

- Think of conditions as “yes or no” questions. They can only be answered by one of two options – “True” or “False”.

```
if condition:
    statement
    statement
    statement
```
Boolean Expressions

- Evaluate to a boolean value

- The data type is called "bool"

- Possible values are True or False

- True and False are boolean literals

- General Note: we can refer to an expression that evaluates to a given type as an expression of that type
  - "integer expression"
  - "float expression"
  - Etc.
Boolean

- 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”

- In 1937 Claude Shannon recognized that Boolean algebra could describe the properties of switching circuits: this is the foundational principle for digital computing
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

\[ a > b \quad \# \text{ is a greater than } b \? \]
\[ a < b \quad \# \text{ is a less than } b \? \]
\[ a == b \quad \# \text{ is a equal to } b \? \]
\[ a <= b \quad \# \text{ is a less than OR} \]
\[ \quad \# \text{ equal to } b \? \]
\[ a >= b \quad \# \text{ is a greater than OR} \]
\[ \quad \# \text{ equal to } b \? \]
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:
    print ('the pen is mightier than the sword!')
    # 10 > 7
    # True
Let’s Evaluate!

# given these variables

```python
a = 99
b = 7
c = -5
d = 92
```

# evaluate these expressions

```python
a > b
b < c
b >= c
b <= d
c <= d
a == b + d
d <= a + c
c != b
```
Comparison Operator Tips

- Don’t confuse "==" with "="
  - "=" is used for assigning values to variables
  - "==" is used for testing to see if two values are identical
  - This is an extremely common source of logic errors!

- 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
Let’s write some programs!
Programming Challenge: Freezing / Boiling 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 two messages based on the information provided:
  - You’re going to freeze your guppy!
  - You’re going to boil your guppy!
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

1. Start
2. Input monthly sales
3. Met quota?
   - True: Assign bonus of $500
   - False: Print "You made your quota!"
4. Finish
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?
The IF – ELSE structure
Simple Selection Statements

- 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?

False
print "Sorry, you didn't qualify"

True
Qualify user for a loan

print "You qualified!"

Finish
The IF-ELSE structure

```python
if temperature < 32:
    print ("it’s freezing outside!")
else:
    print ("it’s not so bad outside …")
```
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 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
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>Character</th>
<th>Code</th>
<th></th>
<th>Character</th>
<th>Code</th>
<th></th>
<th>Character</th>
<th>Code</th>
<th></th>
<th>Character</th>
<th>Code</th>
<th></th>
<th>Character</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>16</td>
<td>16</td>
<td>DLE</td>
<td>33</td>
<td>32</td>
<td>SP</td>
<td>48</td>
<td>0</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>17</td>
<td>17</td>
<td>DC1</td>
<td>33</td>
<td>17</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>18</td>
<td>18</td>
<td>DC2</td>
<td>34</td>
<td>18</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>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>19</td>
<td>19</td>
<td>DC3</td>
<td>35</td>
<td>19</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>20</td>
<td>20</td>
<td>DC4</td>
<td>36</td>
<td>20</td>
<td>$</td>
<td>52</td>
<td>4</td>
<td>68</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>21</td>
<td>21</td>
<td>NAK</td>
<td>37</td>
<td>21</td>
<td>%</td>
<td>53</td>
<td>5</td>
<td>69</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>22</td>
<td>22</td>
<td>SYN</td>
<td>38</td>
<td>22</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>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>23</td>
<td>23</td>
<td>ETB</td>
<td>39</td>
<td>23</td>
<td>'</td>
<td>55</td>
<td>7</td>
<td>71</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>24</td>
<td>24</td>
<td>CAN</td>
<td>40</td>
<td>24</td>
<td>(</td>
<td>56</td>
<td>8</td>
<td>72</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>25</td>
<td>25</td>
<td>EM</td>
<td>41</td>
<td>25</td>
<td>)</td>
<td>57</td>
<td>9</td>
<td>73</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>26</td>
<td>26</td>
<td>SUB</td>
<td>42</td>
<td>26</td>
<td>*</td>
<td>58</td>
<td>58</td>
<td>:)</td>
<td>74</td>
<td>90</td>
<td>Z</td>
<td>106</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>27</td>
<td>27</td>
<td>ESC</td>
<td>43</td>
<td>27</td>
<td>+</td>
<td>59</td>
<td>59</td>
<td>;</td>
<td>75</td>
<td>91</td>
<td>]</td>
<td>107</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>28</td>
<td>28</td>
<td>FS</td>
<td>44</td>
<td>28</td>
<td>,</td>
<td>60</td>
<td>59</td>
<td>&lt;</td>
<td>76</td>
<td>92</td>
<td>\</td>
<td>108</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>29</td>
<td>29</td>
<td>GS</td>
<td>45</td>
<td>29</td>
<td>-</td>
<td>61</td>
<td>60</td>
<td>=</td>
<td>77</td>
<td>93</td>
<td>}</td>
<td>109</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>30</td>
<td>30</td>
<td>RS</td>
<td>46</td>
<td>30</td>
<td>.</td>
<td>62</td>
<td>62</td>
<td>&gt;</td>
<td>78</td>
<td>94</td>
<td>^</td>
<td>110</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>31</td>
<td>31</td>
<td>US</td>
<td>47</td>
<td>31</td>
<td>/</td>
<td>63</td>
<td>63</td>
<td>?</td>
<td>79</td>
<td>95</td>
<td>_</td>
<td>111</td>
</tr>
</tbody>
</table>
Comparison 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”

- Example:
  - string_lc = str.lower(‘Harry Potter’)  # string_lc = ‘harry potter’
  - string_uc = str.upper(‘Harry Potter’)  # string_uc = ‘HARRY POTTER’
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)
Programming Challenge: Alphabetize two strings

- Ask the user to type in two names
- Compare the names and print them out in alphabetical order
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
```
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
Nested Decision Structures
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

→

Number == 5 ?

True → print "You guessed the number!"

False → Number < 5 ?

True → Finish

False → print "too high!"

→

Number < 5 ?

True → Finish

False → print "too low!"
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")
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!
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, an 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’.
Nested Decision Structures

- Indentation is key – Python will use the indentation level of a structure to determine its relationship to any previous statements

- Remember: indentation specifies a block of code
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

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')
```
You can simplify complex IF statements by using the ELIF structure.

ELIF can be used in place of:
```python
else:
    if (cond):
```

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.
g = float(input('grade '))

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.
Write This With IF-ELIF-ELSE

Start

Get Temp.

Temp > hi_thresh?

YES → Turn on A/C

NO

Temp < lo_thresh?

YES → Turn on Heat

NO

Turn off whatever is on.
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.
- 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
  - True and True => True
  - True and False => False
  - False and True => False
  - False and False => False

- The resulting boolean expression will evaluate to be True if the two boolean expressions it is connecting both evaluate to True
Let’s evaluate!

```python
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
Loan Qualifier

```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 the 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
Guppy Temperature

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.
"not" example

username = input('username? ')

if not (username == 'Harry'):
    print("invalid input!")
else:
    print("Welcome, Harry!")
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