LESSON 15

NESTED IF STATEMENTS AND STRING/INTEGER CONVERSION

OBJECTIVE

Learn to work with multiple criteria if statements in decision making programs as well as how to specify strings versus integers in Python.

MATERIALS

This lesson will use the equipment set-up you assembled in Lesson 9 including the Raspberry Pi, monitor, keyboard, and mouse. No internet connection or other materials are needed.

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- Program Flow, Strings, Variables, Print Commands, Order (Lesson 11)
- Code Organization, User Input (Lesson 12)
- Math Functions (Lesson 13)
- Boolean Logic, If/Else Statements (Lesson 14)

LESSON

In this lesson you'll continue to learn how to work with if statements by adding increasingly complex criteria. You will then move on to learn how to specify strings versus integers in Python.

NESTED IF STATEMENTS

In Lesson 14, you learned to structure simple if statements in Python. But what happens if you need to have multiple criteria? For example, if you only wanted to print a receipt if the sale value was $1 and the item was red? For this, you would need to use nested if statements.

A nested if statement, describes an if statement that's inside of, and dependent on, another if statement. This can be used to add extra layers of decisions that you might want to make happen in your program.

FORMATTING: IMPORTANCE OF INDENTATION

When formatting nested if statements, it's important to indent the code. This tells Python that the statements are dependent upon each other.

For example, this program simulates a code that might open a lock. The lock can only open with the code 345.

```
if x == 3:
    print('First digit accepted')
    if y == 4:
        print('Second digit accepted')
        if z == 5:
            print('Third digit Accepted')
            print('Lock open')
```

What will happen when the digits 3, 4, 5 are entered?

The code digits are initially programmed in using the variables x, y, and z. The first if condition x ==3 will be evaluated, and if TRUE, **First digit accepted** will be printed, and the next condition will be evaluated. If y == 4 evaluates as TRUE then **Second digit**

accepted will be printed, and the third condition z == 5 will be evaluated. If this evaluates as TRUE, then **Third Digit accepted** and **Lock open** will both be printed.

The only way **Lock Open** can ever be printed is if all three conditions preceding it evaluate as TRUE. If any of the digits do not equal the expected digit, then the rest of the code will be skipped, and the lock will not open.

But what happens if the statements aren't properly indented? Python no longer sees them as dependent upon one another and you can't obtain the conditional buildup seen in the last example.

For example, consider the following code:

```
if x == 3:
    print('First digit accepted')
if y == 4:
    print('Second digit accepted')
if z == 5:
    print('Third digit Accepted')
    print('Lock open')
```

What happens if the user enters 555?

Since the first digit is wrong, the first condition will evaluate as FALSE, so no message would be printed. Due to the improper indentation, the second condition will be evaluated, even though the first digit was incorrect. Without indentation, Python doesn't realize it's not supposed to proceed to the next line of code! In this case, the second digit is also incorrect, so no message will be printed.

The problem really shows up when the third condition is tested. This digit is correct, so the condition will evaluate as TRUE, and both **Third digit accepted** and **Lock open** will be printed. The lock was allowed to open, even though two of the three digits were incorrect. Without indentation, Python treated all three conditions as independent of one another.

If your intent is to have multiple if statements trigger an action, then make sure your indentation is allowing the conditions to be evaluated as expected. Also, test to make sure your program is operating as designed, and that incorrect entries don't generate unexpected behavior.

ALWAYS USE FOUR SPACES TO INDENT CODE. It may be tempting to use the tab key out of convenience, but tabs can be interpreted differently by different programs. Spaces are treated the same way in every program, so they are the safest way to learn to indent your code. If you need more levels of indentation just add four more spaces for each level:

```
first_line                  # this line has no spaces
    second_line             # this line has 4 spaces
        third_line          # this line has 8 spaces
    fourth_line             # this line has 4 spaces
        fifth_line          # this line has 8 spaces
```

STRINGS VERSUS INTEGERS

As you learned in the previous lesson, strings are one or more characters like letters, numbers, or symbols. You can ask Python if string '6' equals string '6' and it will evaluate as TRUE. This happens because those two characters are the same, much like Python would say that the strings 'a' and 'a' are equal. You can also add string 'a' to string 'a' and you will get an output string of 'aa'. This worked great in previous lessons to add “My name is “ and “name” together to get a full sentence.

The problem occurs when you try to do mathematical calculations with strings of numbers. Consider the following program:

```
a = '4'

b = '2'

c = a + b

print(c)
```

What number do you think will be printed to the console as c? Here is a hint: it's not 6.

42 will be printed because the character '4' added to the character '2' is '42'. Python doesn't understand that these characters represent numeric values. For that you will need to use integers.

CONVERTING A VALUE TO AN INTEGER

Integers are only allowed to be numbers, so trying to store the value 'apple' as an integer will result in an error. Here are some examples:

Statement	**Action**	**Use for numeric functions?**
x = '6'	This will store the value 6 as a string in a variable called x	No
x = 6	This will store the value 6 as an integer in a variable called x	Yes
x = input('Type a number: ')	Typing 6 at this prompt will cause the program to store the value of 6 as a string in a variable called x	No

The last one is a bit of a problem. How do you use an input of 6 for a mathematical equation when Python is storing it as a strong? Fortunately, Python has a great way to convert back and forth between strings and integers. The command for converting a numeric string to an integer is:

```
int()
```

Just insert a variable value between the parentheses, and Python will convert the value to an integer, so you can use that value for math purposes. Remember, this will only work on numeric values. Attempting to convert the string value 'apple' into an integer will result in a Python error.

Here is an example of converting a string to an integer:

`x = '42'`	This will initially store the value of 42 as a string in variable x
`y = int(x)`	This will take the current value of x, convert it to an integer, and store that new value in the variable y

This integer command will be very useful in Activity #3, where you will convert the numeric user input from a string into an integer, so you can compare it to other numbers later in the program.

CONVERTING A VALUE TO A STRING

But what if you have the opposite problem? You have an integer you need to convert to a string so you can combine it with another string? Python cannot merge integers and strings to be displayed together so you must convert any integers to strings.

The command to convert a value to a string is:

`str()`

It can be used just like the **`int()`** command above. Here is an example of converting an integer to a string:

`x = 42`	This will initially store the value of 42 as an integer in variable x
`y = str(x)`	This will take the current value of x, convert it to a string, and store that new value in the variable y

This can be helpful when trying to print integers and strings together. As noted above, Python cannot concatenate (merge) strings and integers together in the same print command.

Bad Code	Good Code

`x = 42`
`print('Your number is ' + x)`
 	
This print statement is trying to combine the string 'Your number is ' and the integer value of x. This will result in an error.
	`x = 42`
	`y = str(x)`
	
	`print('Your number is ' + y)`
	
	Since x is being converted into a string value called y, this print statement can combine 'Your number is ' and the string value of y, with no errors. 'Your number is 42' will be printed to the shell.

This conversion can also happen within a single line. In Lesson 13 you used this statement to bring in user input as an integer:

```
var = int(input('Enter a number: '))
```

Normally the input command will bring in any value entered as a string, even numeric values. The problem with this is that you can't perform math operations on a string, even when the string is something like '6'. That '6' might as well be 'apple' or 'tree'. Python has no idea how to subtract 2 from 'apple' or from '6'.

The input string must be converted to an integer using the **int()** command. There are a couple of ways to accomplish the same thing:

Option #1:

```
var = input('Enter a number: ')

var = int(var)
```

The input from the user is initially saved to the variable **var**. Then the int(var) will take the integer value **var** and overwrite the string value of **var** with the integer value.

Or you can do this by wrapping the **int()** around the entire input command as seen below.

Option #2:

```
var = int(input('Enter a number: '))
```

In this example, from Lesson 13, the user is asked for input, that input is converted to an integer, and that integer value is saved to the variable called **var**. This way is slightly more complicated looking, but it results in one less line of code.

ACTIVITIES

In the activities for this lesson you will practice using nested if statements to evaluate user input as well as practice converting strings to variables and variables to strings in Python.

ACTIVITY #1 – ADD FIVE YEARS TO YOUR AGE

In this activity you will write a program that will ask the user for their current age, add five years, and print a message letting them know what their age will be in five years.

STEP #1

Ask the user how old they are and store it in a variable called **age**:

```
age = input('How old are you: ')
```

STEP #2

The users age is now stored as a string in a variable called age, however Python can't add five to a string. The string value must be converted to an integer, so Python can add to the value. Store that new integer value as a variable called **age_int**:

```
age_int = int(age)
```

STEP #3

Now Python can add five years to the current value of **age_int**:

```
age_int = age_int + 5
```

This will take the current value of **age_int**, add 5 to it, and save it as the updated value of **age_int**.

STEP #4

You want to have a nice message that combines the string **Your age in five years will be** and the **age_int** value. Since the **age_int** value is currently an integer is cannot be combined with the other string. You must convert the **age_int** value from an integer back to a string. Save this new string as **age_str**:

```
age_str = str(age_int)
```

STEP #5

You now have two strings that can be combined into a single **print** command. Combine the string **Your age in five years** will be and the **age_str** value into a **print** command:

```
print('Your age in five years will be ' + age_str)
```

STEP #6

Verify the fully assembled program is free of errors:

```
age = input('How old are you: ')

age_int = int(age)



age_int = age_int + 5



age_str = str(age_int)

print('Your age in five years will be ' + age_str)
```

STEP #7

Run the program a few times using different age values and make sure that you get the expected output each time.

Optional: Save the program if desired.

ACTIVITY #2 – AGE CALCULATOR

In this activity you will write a program that will ask a user for the year they were born and if their birthday has occurred yet this year. The program will then calculate the user's age at the end of this year and will subtract a year if their birthday has not yet occurred. The program will then display their current age in a single print statement.

STEP #1

Ask the user for the year they were born and store it as a variable called **year**:

```
year = input('What year were you born: ')
```

STEP #2

Ask the user if their birthday has happened yet this year and store it as a variable called **bday**:

```
bday = input('Has your birthday happened this year (y/n): ')
```

STEP #3

Now, convert the year from a string to an integer using the **int()** command and store it as a variable called **year_int**:

```
year_int = int(year)
```

STEP #4

The **year_int** is now storing the integer value of the year the user was born. You can use this value for math purposes. Subtract that year from the current year and store the result as a variable called **age**:

```
age = 2018 – year_int
```

STEP #5

The **age** value is currently the user's age at the end of this year. If their birthday has not yet occurred, then they will be a year younger than the age value. Use an if condition to see if the value of **bday** is equal to **'n'**, and if so, subtract 1 from the value of **age**. Remember to end the if statement with a colon and to indent the code to be executed by the if statement:

```
if bday == 'n':

    age = age - 1
```

If the user answered **'n'** then 1 year will be subtracted from the value of **age**. You don't need an else statement because if the user answered **'y'** then **age** does not need to be modified.

STEP #6

The **age** value is currently an integer, so it cannot be easily printed with a string. Convert **age** to its string equivalent using the **str()** command, and use a variable called **age_str** to store the string value:

```
age_str = str(age)
```

STEP #7

Build a **print** statement that will bring everything together:

```
print('Your current age is ' + age_str)
```

STEP #8

Verify the fully assembled program is free of errors:

```
year = input('What year were you born: ')

bday = input('Has your birthday happened this year (y/n): ')


year_int = int(year)

age = 2018 - year_int


if bday == 'n':

    age = age - 1


age_str = str(age)


print('Your current age is ' + age_str)
```

STEP #9

Run your program a few times to make sure the age it's outputting is correct, and that changing whether or not your birthday happened this year gives you the correct age.

Optional: Save the program if desired.

ACTIVITY #3 – GUESS A NUMBER

You will now write a program that will ask you to guess a number (hint: the number is 6). If you guess too low, too high, or just right, the program will let you know.

STEP #1

Ask the user to input a number between 1 and 10 and assign it to a variable called **guess**:

```
guess = input('Pick a number between 1 and 10: ')
```

STEP #2

Now that you have the user input stored as a string named **guess**, convert it to an integer, so you can compare it to other numeric values, to determine if the guess is too low, too high, or the correct number:

```
guess_int = int(guess)
```

This will take the string value of **guess**, convert it to an integer value, and store it back in a variable called **guess_int**.

STEP #3

Compare the numeric value of **guess_int** to some different criteria. Use the first **if** condition to check if the guess is too low:

```
if guess_int < 6:

  print('Too low, better luck next time!')
```

This will check to see if the number is less than 6. If so, it will print the message in the print statement.

STEP #4

Use an **elif** condition to check if the guess was greater than 6:

```
elif guess_int > 6:

  print('Too high, better luck next time!')
```

STEP #5

Add the **elif** condition that will trigger if the user correctly guesses the number 6:

```
elif guess_int == 6:

  print('Great guess,  you are correct!')
```

STEP #6

Add the final **else** condition that will display an error if the user's guess does not trigger any of the previous statements:

```
else:

  print('Invalid input, please try again')
```

STEP #7

There's a problem with the code above. Can you spot it?

What if the user guesses 42? 42 is greater than 6 so the "Too high" print statement will be triggered. The rules only allow for guesses between 1 and 10, so you need to limit the conditions to be more specific. Fix the less than 6 condition first. You need to allow for guesses between 1 and 5, so anything greater than 0 or anything less than 6. That would look like this:

```
if guess_int > 0 and guess_int < 6:
```

This will work but you can combine the two conditions to make one shorter statement:

```
if 0 < guess_int < 6:
```

This condition will only evaluate as true for guesses between 0 and 5. We can apply this same logic to the greater than 6 condition to limit it guesses that are greater than 6 but less than or equal to 10:

```
elif 6 < guess < = 10:
```

Now if a user guesses 42 it will no longer trigger the **`'Too high'`** condition. That guess will trigger the else statement and print **`"Invalid input, please try again"`**.

STEP #8

Verify the fully assembled program is free of errors:

```
guess = input('Pick a number between 1 and 10: ')

guess_int = int(guess)



if 0 < guess_int < 6:

  print('Too low, better luck next time!')

elif 6 < guess_int <= 10:

  print('Too high, better luck next time!')

elif guess_int == 6:

  print('Great guess,  you are correct!')

else:

  print('Invalid input, please try again')
```

STEP #9

Try running the program and make a bunch of different guesses to ensure you get the expected output from each guess.

Optional: Save the program if desired.

QUESTIONS FOR UNDERSTANDING

1. Can Python print strings and integers together in the same print statement?

2. Is the indentation optional for nested if statements?

3. Can you modify the program from Activity #1 to give the user their age in 20 years?

4. Can you modify the program from Activity #3 to allow guesses from 1 to 100 and change the right answer to 42?

Answers can be found on the next page.

ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

1. *Can Python print strings and integers together in the same print statement?*

   ANSWER: No, strings and integers cannot be used together in the same print statement. You can only have all strings or all integers in a print statement. Since most print statements have letters or words included, it is generally necessary to convert integers to strings so they can be included in the print statement.

2. *Is the indentation optional for nested if statements?*

   ANSWER: No. Lack of indentation removes the conditional (nested) nature of the statements. Without indentation, the statements will each evaluate individually without relying on the conditions of the previous statements.

3. *Can you modify the program from activity #1 to give the user their age in 20 years?*

   ANSWER: Yes, you can modify the code as follows (the change to the code is highlighted for your convenience):

```
age = input('How old are you: ')

age_int = int(age)


age_int = age_int + 20


age_str = str(age_int)

print('Your age in twenty years will be ' + age_str)
```

4. *Can you modify the program from Activity #3 to allow guesses from 1 to 100 and change the right answer to 42?*

   ANSWER: Yes, you can modify the code as follows (the change to the code is highlighted for your convenience):

```
guess = input('Pick a number between 1 and 100: ')
guess_int = int(guess)


if 0 < guess_int < 42:
  print('Too low, better luck next time!')
elif 42 < guess_int <= 100:
  print('Too high, better luck next time!')
elif guess_int == 42:
  print('Great guess,  you are correct!')
else:
  print('Invalid input, please try again')
```

CONCLUSION

In this lesson you learned to properly format nested if statements to allow for conditional criteria in your logic statements. You also learned to convert strings to integers for calculation purposes, and integers to strings to allow the characters to be combined with other strings in print statements, etc.

In the next lesson, you will brush off your electronic skills and learn to connect a breadboard circuit to the Raspberry Pi and write a program in Python to control the circuit.