(Python) Chapter 2: If Statement, Random Class, Introduction to Defining Functions

2.1 Conditional Execution

Basic Idea

One limitation to programs created only using the statements presented in chapter 1 is that the same exact statements in a program will run every time the program is interpreted. The problem with this is that in real life, when we carry out directions, we don't always execute the same steps. Consider the situation of determining whether or not you will go out with a friend. If your homework is done, you would like to go out with your friend. But, if your homework isn't done, you won't go out with your friend. Similarly, in programming, it makes sense to allow conditional execution. Namely, if some condition is true, then execute some set of statements.

Basic if Statement

In Python, the syntax of the most basic if statement is as follows:

```
if <Boolean Expression>:
    stmt1
    stmt2
    ...
    stmtn
stmtA
```

A Boolean expression is one that always evaluates to true or false. Details about how to create a Boolean expression will be covered shortly. If this expression evaluates to true, then the statements stmt1 through stmtn are executed in order, followed by stmtA. However, if this expression evaluates to false, then all of these statements are skipped and stmtA is then executed. Note: It's not required for there to be a statement such as stmtA after the completion of the if statement.

The interpreter determines which statements are inside of the if clause based on indentation. For a statement to be consider inside of the if, it must be indented to the right from the if statement itself. All subsequent statements inside of the if must be indented to the same level.

Sales Tax Example Revisited

When buying most items, sales tax is added to the price. However, for some items, such as basic food, no sales tax is added. In this example we'll ask the user to enter the item price. Then we'll ask them if sales tax is to be assessed. If it is, then we'll ask the for percentage of sales tax and calculate the final price.

```
# Arup Guha
# 6/26/2012
# Sales Tax Program Revisited - conditionally charges sales tax.

def main():

    # Get the user input.
    item_price = float(input("Please enter the price of your item.\n"))
    is_taxed = input("Is your item taxed(yes,no)?\n")

    # If the item is taxed, ask the sales tax percentage and add tax.
    if is_taxed == "yes":
        tax_rate = float(input("What is the sales tax percentage?\n"))
        item_price = item_price + item_price*tax_rate/100

    # Calculate the total price and round.
    print("Your total cost is $",item_price,".",sep="")

# Start the program.
main()
```

This program shows our first example of a Boolean expression. The Boolean expression in this program is:

```
is_taxed == "yes"
```

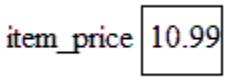
This is how we check to see if the variable is_taxed is equal to the string "yes". If it is, then this Boolean expression evaluates to true. Otherwise, it evaluates to false.

Thus, if the user enters "yes", then they will be prompted to enter the percentage of sales tax. Then the variable item_price will be reassigned to include sales tax. If the user enters anything but "yes", then these two statements are skipped. Afterwards, the value of the variable item_price is printed.

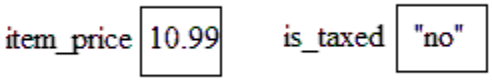
Let's take a look of running this program two separate times:

```
>>>
Please enter the price of your item.
10.99
Is your item taxed(yes,no)?
no
Your total cost is $10.99.
```

After the first line, the picture in memory is as follows:



After the second line, the picture in memory is:



At this point, we approach the if statement. We compare the value of the variable is_taxed to the string literal "yes", and see that they are not equal. Note that when we type in strings we don't type in the double quotes, but when we denote string literals (string values instead of string variables) inside of our programs, we denote them with either matching double quotes or matching single quotes, as was previously discussed in the section about the print statement.

Since this if statement evaluates to false, the following statements that are indented get skipped. The next statement that runs is:

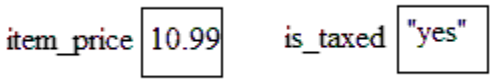
```
print("Your total cost is $",item_price,".",sep="")
```

Since the value of the variable item_price is 10.99 at this point in time, this is what gets printed for the total cost.

Now, consider the following execution of the program:

```
>>>
Please enter the price of your item.
10.99
Is your item taxed(yes,no)?
yes
What is the sales tax percentage?
6.5
Your total cost is $11.70435.
```

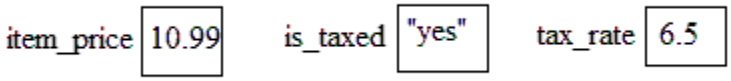
The picture for this execution after the first two lines of code is:



At this point, when we evaluate the Boolean expression in the if statement, we find that it's true since the variable is_taxed stores the string "yes". Then we go ahead and execute the following statement:

```
tax_rate = float(input("What is the sales tax percentage?\n"))
```

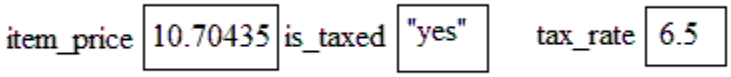
After this statement is executed, our picture of memory is as follows:



Then we execute the following statement in the if:

```
item_price = item_price + item_price*tax_rate/100
```

item_price currently evaluates to 10.99 while item_price*tax_rate/100 is equals to .71435. Adding these, we evaluate the right-hand side of the assignment statement to equal 11.70435, thus our picture in memory AFTER this statement is:



One of the basic building blocks of a Boolean expression is a relational operator. Here are the six relational operators and their meanings:

Relational Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Thus, in this Boolean expression we are checking to see **IF** the variable is_taxed has the value "yes". Notice that checking for equality uses two equal signs instead of one. This is because one

equal sign already has a well-defined meaning: the assignment operator. Assigning a variable changes its value while checking for equality between two expressions doesn't change the value of any of the variables involved.

Formatting Decimal Output to a Specific Number of Places

In our previous examples, when we printed out real numbers, they printed out to many decimal places. Python uses a method similar to the language C to format real numbers to a fixed number of decimals. The syntax is strange and uses that percent sign (%), which we use for mod, in a different way. The expression that evaluates to a variable rounded to two decimal places is:

```
"%.2f"%var
```

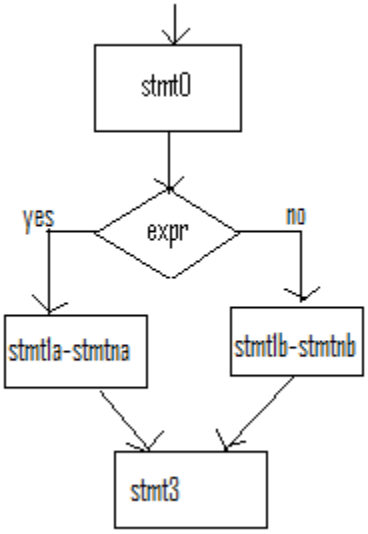
where var is the variable to round. Here is an application of this syntax to displaying the item price rounded:

```
print("Your total cost is $", "%.2f"%item_price, ".",sep="")
```

If you want a different number of decimal places displayed, just change the 2 in the format string. (Note: The f in that set of double quotes indicates float and the .2 indicates to print out two digits after the decimal.)

2.2 if Statement with else Clause

In the previous example, if the item was taxed, we wanted to carry out an action, but if it wasn't we simply wanted to skip that action. In many cases however, if some condition is true, we want to execute one set of statements, but if it's false, we want to execute a separate set of statements.



The basic syntax for this type of situation is as follows:

```
if <Boolean Expression>:
    stmt1a
    stmt2a
    ...
    stmtna
else:
    stmt1b
    stmt2b
    ...
    stmtmb
stmtA
```

The basic flow of control here is that we first evaluate the Boolean expression. If it's true, we complete statements stmt1a through stmtna and then continue to stmtA. Alternatively, if the Boolean expression is false, skip stmt1a through stmtna, but do execute statements stmt1b through stmtmb, and then continue to stmtA.

Let's take a look at a couple examples that utilize this component of the if statement.

Work Example

Consider a job with flexible hours where you must spend a certain number of hours a week. During the week if you've exceeded that number, let's say you have to take the excess hours as vacation in future weeks. Alternatively, if you haven't exceeded that number, you'll have to work the remainder of the hours. In this program, we will ask the user to enter the number of hours they are supposed to work a week and how many she's worked thus far. Then, our program will print the appropriate output, asking the user to either work more hours, or take vacation.

```
# Arup Guha
# 7/2/2012
# Example of a Basic if-else statment - determines if you need to
# work more or if you need to take vacation time.

def main():

    work_week = int(input("How many hours are you supposed to work?\n"))
    this_week = int(input("How many hours have you worked this week?\n"))

    # You've worked enough!
    if this_week > work_week:
        print("You must take",this_week-work_week,"hours of vacation.")

    # Need to put in some more hours!!!
    else:
        print("You must still work",work_week-this_week,"hours this week.")

main()
```

Now, in the case that the Boolean expression is true, we print out the vacation hours. Alternatively, we print out the hours left to work. Incidentally, what happens if you've worked the exact correct number of hours?

Quadratic Equation Example

A common formula taught in Algebra I is the quadratic formula. However, sometimes this formula leads to "impossible" roots, which we later learn are "complex." In this program, given the coefficients of a quadratic equation from the user, if the roots are real, we will print them out. If they are not, we'll print out an error message.

The quadratic formula is as follows: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. This equation has to real roots so long as what is under the square root sign is non-negative. This leads to the following program:

```
# Arup Guha
# 7/2/2012
# Quadratic Equation Solver

def main():

    # Get user input.
    a = float(input("Please enter a from your quadratic equation.\n"))
    b = float(input("Please enter b from your quadratic equation.\n"))
    c = float(input("Please enter c from your quadratic equation.\n"))

    # Calculate the discriminant.
    disc = b**2 - 4*a*c

    # Deal with real roots.
    if disc >= 0:

        x1 = (-b + disc**.5)/(2*a)
        x2 = (-b - disc**.5)/(2*a)

        print("Your roots are ",x1," and ",x2,".", sep="")

    # Error message for complex roots.
    else:

        print("Sorry, your roots are complex.")

main()
```

elif clause

In the work week example, if the user worked the exact correct number of hours, our program would print the following message:

```
You must still work 0 hours this week.
```

While this is technically accurate, the tone of this message is a bit misleading. It would be nice if we had a third "option" to print out in this special equal case.

Luckily, python gives us the facility to check for 3 or even more different options and choose at most one of them. This is through the elif branch of the if statement. elif is short for "else if." The general syntax of an if statement with one of these branches is as follows:

```
if <Boolean Expression 1>:
    stmt1a
    ...
    stmtna
elif <Boolean Expression 2>:
    stmt1b
    ...
    stmtmb
else:
    stmt1c
    ...
    stmtpc
stmtA
```

This works as follows: We first check the first Boolean expression. If it's true, we do stmt1a through stmtna, and then skip to stmtA. Alternatively, if this is false, we then check the second Boolean expression. If this one's true, then we execute stmt1b thorugh stmtmb and then skip to stmtA. Finally, if the second Boolean expression is also false, we go to the else clause and execute statemnts stmt1c through stmtpc and then move onto stmtA.

Thus, we can edit the if statement in our work program as follows:

```
    # You've worked enough!
    if this_week > work_week:
        print("You must take",this_week-work_week,"hours of vacation.")

    # Correct hours worked
    elif this_week == work_week:
        print("Perfect, your done for work for the week!")

    # Need to put in some more hours!!!
    else:
        print("You must still work",work_week-this_week,"hours this week.")
```

Grade Example

A very common example given to illustrate an if statement with several clauses is a program that prints out the grade a student should get based on the percentage they earned in a class. In this example, we'll use the typical A (90-100), B (80-89), C(70-79), D(60-69) and F (0-59) breakdown.

```
def main():

    perc = int(input("What is your percentage in class?\n"))

    if perc >= 90:
        print("You got an A!")
    elif perc >= 80:
        print("You got a B!")
    elif perc >= 70:
        print("You got a C.")
    elif perc >= 60:
        print("You got a D.")
    else:
        print("Sorry, you got a F.")

main()
```

Notice that we only need to check one condition for each letter grade since the order in which they are checked. All grades 90 or higher are "caught" by the first clause, so if the second clause (elif perc >= 80) is ever evaluated, then we know that perc must be less than 90. Thus, if this boolean expession is true, it follows that perc is in greater than or equal to 80 AND less than 90. Continuing this logic, each of the first four clauses properly maps to their corresponding letter ranges. The only way the else clause executes is if perc is less than 60.

To note that the order here is important, consider what would happen with the following if statement:

```
    if perc >= 70:
        print("You got an C.")
    elif perc >= 90:
        print("You got a A!")
    elif perc >= 80:
        print("You got a B!")
    elif perc >= 60:
        print("You got a D.")
    else:
        print("Sorry, you got a F.")
```

What would this code segment print out if perc equals 95 right before it? Or 83? Will this code segment ever print out "A" or "B"?