IF STATEMENTS

As we have already seen, a computer program is a list of instructions. In what order are the instructions performed? By default, in the order that we specify. In other words, if a computer program has 3 statements, then the computer will perform the first statement, followed by the second statement, and finally the third statement.

To be able to solve more useful problems, we need to write computer programs that can do more than just sequential execution. In particular, two capabilities are needed:

- To make choices; to take a fork in the road. In other words, to follow one path instead of another. This includes the ability to skip some statements that we sometimes don't need to perform.
- To repeat some steps. In other words, sometimes we want a "loop."

This chapter will cover making choices. In the next chapter we will see repetition.

In computer science, we usually use the term "control structures" or "control flow" when referring to these two capabilities. We will practice making choices and repetition quite a lot, because they are so important in problem solving. Along the way, we will learn some Python keywords to handle these situations. The words `if`, `elif` and `else` are used for making choices, and the words `while` and `for` are used for repetition.

Before we dive into our discussion of control structures, I want to review some basic facts about computer programs:

- A program needs to have output. (Otherwise, how would we know that it works correctly?)
- Output is usually the result of some calculations we need to do in the program.
- A program's calculations are usually based on input.

This is why we tend to say that a computer program has 3 important parts: input, calculations and output. We can solve a greater variety of problems if we allow our programs also to *make choices*, to ask questions about either the input values or the results of calculations. A program can decide which path to follow based on whether the answer is yes or no.

It turns out that most programs need to perform different actions depending on the input. For example, let's suppose you were calculating income tax. You would ask the user for an amount of income. A high level of income might incur a 40% income tax. But a low income might result in no income tax at all. Therefore, depending on the income, we may need to use a different formula to compute the tax.

The if-statement

Sometimes, we need our program to make a decision. In other words, we want our program to do something based on what values are in our variables. Here is the basic format of an if-statement:

```
if <condition>:
    # steps to perform if the <condition> is True
```

Optionally, we can include an "else" portion, so that the entire if-statement looks like this:

```
if <condition>:
    # steps to perform if the <condition> is True
else:
    # steps to perform if the <condition> is False
```

Every if-statement has a condition to test. An if-statement is essentially asking a yes/no question, usually about the value in a variable. Comparisons are done with a *relational operator*. There are 6 relational operators that we can use in Python:

Relational operator	Meaning
<	is less than
>	is greater than
<=	is less than or equal to (i.e. is at most)
>=	is greater than or equal to (i.e. is at least)
==	is equal to
!=	is not equal to

Please note the difference between = and ==. The = operator is used in an assignment statement. The == is a relational operator. To illustrate, consider the following examples:

- x = 5 means that we command the computer to put the number 5 into the variable x.
- x == 5 is a question asking the computer if x is equal to 5 or not.

A comparison is an example of a *boolean* expression. Boolean means the value is either `True` or `False`. It turns out that boolean is actually a data type in Python, along with int, float, and str. So now you know four data types. With the boolean type, it is possible to assign a variable to be `True` or `False`. Later, we will see examples where this is especially useful.

Using the words “and” and “or”

When using relational operators to ask a question about a variable, it’s even possible to ask multiple questions, combined with the words `and` or `or`. For example, an `if` statement could perform two comparisons like this: `if answer == 1 or answer == 3:`

This is sometimes called a “compound condition” because it consists of two or more conditions separated by the words `and` or `or`. Here, this first condition is answer == 1, and the second condition is answer == 3. The purpose of the compound condition is to test whether the value of answer is either 1 or 3. Please note that when typing out a compound condition like this, all of the individual conditions that make up the compound condition need to contain a relational operator. For example, one mistake that beginners sometimes make is to write an `if` condition like this:

```
if month == "September" or "April" or "June" or "November":
```

The intent here is to create a compound condition with four parts. However, the string “April” by itself is not a condition. To write this correctly, we have to spell out exactly what we want as 4 complete conditions, as follows.

```
if month == "September" or month == "April" or month == "June" or \
   month == "November":
```

We often need to use the word `and` if we want to ask if a variable is between two values. For example, we might want to know if x is an integer between 50 and 100. We could write the condition as follows:

```
if x > 50 and x < 100:
```

As you know, an inequality expression can be turned around as long as we remember to reverse the inequality sign. So, x > 10 has the same meaning as 10 < x. However, when comparing a variable to a constant, it is customary to put the variable on the left and constant on the right. Nevertheless, it is legal to write the above example as:

```
if 50 < x and x < 100:
```

And as soon as we write it this way, we notice that in ordinary mathematics, there is a more compact way of stating this condition. We can take out the word “and” and write the condition this way:

```
if 50 < x < 100:
```

The good news is that Python also allows you to write conditional expressions like this, if you prefer. Note that if we wanted x to be between 50 and 100 *inclusive*, we would write

```
if 50 <= x <= 100:
```

Here is an example of a program or program fragment that uses if-statements.

```
# input
n = int(input("Enter an integer:  "))

# Tell user if it's positive
if n > 0:
    print("Your number is positive.")
# ------------------------------
# Tell user if the number is odd or even
if n % 2 == 0:
    print("Your number is even.")
else:
    print("Your number is odd.")
```

It is important to take note of some syntax requirements with if-statements.

1. A colon is required at the end of the line introducing the if-statement. You also need a colon at the end of the line containing the word `else`.
2. You need to indent the body of the if-statement, and the body of the else clause. Indentation is required because the body may contain more than one statement. Indentation is Python's way of recognizing what is part of the body, and what is not. As you type your program, your editor might indent automatically for you. If not, hit the space bar 4 times to achieve the proper level of indentation.

You are already familiar with the concept of indentation when grouping information in a hierarchical manner. Think of an outline, or the bullets of a presentation. For example, suppose you are studying the notes of a meteorology lesson. A simple outline might look like this:

1. Temperature
2. Precipitation
    a. Rain
    b. Snow
    c. Hail
3. Wind

Notice that section 2 has subsections a, b and c. When we finish subsection c and want to proceed to section 3, we unindent back to the level of the Arabic numerals. Python code is indented and unindented the same way, except that we do not label our statements with numbers or letters.

Let's practice writing if conditions for the following situations. In the following exercises, assume that we have a positive number held in an integer variable called n. In each case, you should write a line of code that introduces an if-statement. It will have this format, and all you have to do is to supply what goes in the blank.

if _____:

1. The number is between 5 and 25 inclusive.
2. The number is less than 12 or equal to 95.
3. The number is even and less than 100.
4. The number is between 10 and 20, or between 80 and 90.
5. The number ends in 7.
6. The number ends in 67.
7. The number is neither 13 nor 17.
8. Now, let's suppose we have two numbers x and y. Write the condition that says that y is positive and x is strictly between 0 and y.

A very useful application of the if-statement is determining if a year is a leap year. The traditional Julian definition of leap years is pretty simple. We just need to verify that the year number is divisible by 4. Here is what the code would like:

```
if year % 4 == 0:
    print("Leap year")
else:
    print("Not a leap year")
```

Unfortunately, we no longer use the Julian definition of leap years. It turns out that to be more astronomically accurate, we need to use what is called the Gregorian definition of leap years. The rules to qualify as a leap year are as follows:

- If the year ends in 00, then the year must be divisible by 400.
- If the year doesn't end in 00, then it must be divisible by 4.

Exercises:

1. Using the Gregorian definition, write Python code that will print "Leap year" or "Not a leap year", as appropriate, based on the value of the `year` variable.

2. Give an example of a year that qualifies as a leap year under the Julian definition but not under the Gregorian definition.

3. Let's calculate someone's weekly wage. Ask the user for the hourly rate and number of hours worked. Be sure to pay overtime if the number of hours worked is over 40: for each additional hour the hourly rate needs to be 50% higher.

4. Ask the user to enter two numbers. Determine which number is larger, and by how much. For example, if the user enters 2 and 6, tell the user that the second number is larger, and that the difference is 4.

5. Suppose we have an integer variable numDogs. We want to print a sentence such as `"There are 5 dogs."`, but if the value of numDogs is 1, we need to print the word `dog` in the singular. Show how we can handle this.

6. How would you find the absolute value of the difference between two numbers?

7. Explain what is wrong in each of the following code fragments.

```
numPeople = int(input("Enter number of people:  "))
if numPeople = 0:
    print("You entered zero people.")
```

```
-------------------------------------------------
if c == 'a' or 'e' or 'i' or 'o' or 'u':
    print("The letter is a vowel.")
```

```
-------------------------------------------------
choice = input("Do you want to quit the game?  (y/n):  ")
if choice != 'Y' or choice != 'y':
    print("Great!  The game will continue...")
```

<u>If-statements using multiple choices</u>

Just as it's possible for a program to need several variables, it's also possible that we need more than 2 alternatives with our if-statements. How would we handle 3 or more? For example, given an input value, we may need to test if it is positive, negative or zero. Another example is converting a numerical grade to one of 5 different letter grades according to a grading scale.

You already know that we use the words `if` and `else` with if-statements. The way to allow for additional choices is by using the Python keyword `elif`, which is an abbreviation for "else if." Here is the general format of an if-statement that uses elif:

if <condition 1>:
  # steps to perform if condition 1 is true
elif <condition 2>:
  # steps to perform if condition 2 is true
# You can have as many `elif`s as needed
else:
  # steps to perform if all conditions were false

For example, here is how to test for positive / negative / zero:

```
if value > 0:
    # do positive case

elif value < 0:
    # do negative case

else:
    # At this point, we know value has to be zero!
```

And here is how a grading scale can be used to assign letter grades. If the criteria are 90, 80, 70 and 60, the code would look like this:

```
if grade >= 90:
    letter = 'A'
elif grade >= 80:
    letter = 'B'
elif grade >= 70:
    letter = 'C'
elif grade >= 60:
    letter = 'D'
else:
    letter = 'F'
```

Please note that when you write an if-statement with several alternatives, only one of the bodies can execute. Suppose that grade equals 75. Even though 75 > 70 and 75 > 60, the value of letter is 'C' not 'D', because we execute the body corresponding to the first condition that we encounter that is true. Once we execute the body of the `if` or one of the `elif` conditions, we exit the entire if-statement.

The above grading scale can be adapted to other purposes as well. For example, hurricanes and tornadoes are often classified by a five-category scale according to their maximum wind speed (Saffir-Simpson for hurricanes and Fujita for tornadoes).

Discussion: Modify the grading scale code above to account for plus and minus grades. For example, let's assume that grades ending in 0 or 1 get a minus, and grades ending in 8 or 9 get a plus. What other assumptions should you make?

Here are two complete programs that feature if statements.

```
# isTriangle.py - Given the lengths of 3 line segments, see if they can
# form a triangle.  The rule is that the longest has to be shorter than the
# sum of the other two sides.  But first, we must determine which line
# segment is the longest.  This program illustrates the if statement.

# Input
print("Let's see if you really have a triangle.")
a = float(input("Enter side 1:  "))
b = float(input("Enter side 2:  "))
c = float(input("Enter side 3:  "))

# Calculations
# We first determine which side is the longest, and call it longest.
# The other two sides will be called leg1 and leg2.

if a > b and a > c:
    longest = a
    leg1 = b
    leg2 = c
elif b > a and b > c:
    longest = b
    leg1 = a
    leg2 = c
else:
    longest = c
    leg1 = a
    leg2 = b

# Test for being a real triangle.
if longest < leg1 + leg2:
    print("Yes, this is a triangle.")
else:
    print("No, the three sides cannot form a triangle.")
```

```
# treadmill.py - Determine how many calories are burned on the treadmill.
# We need the following numbers from the user:
# Speed, incline, time, weight
# Formula comes from
# https://www.livestrong.com/article/34973-calculate-treadmill-calories/

# input
pounds = float(input("Enter your weight in pounds:  "))
mph = float(input("How fast did you go (mph)?  "))
incline = float(input("What was the % incline?  "))
minutes = float(input("How many minutes did you walk/run?  "))


# Calculations
# Let's begin with the necessary unit conversions.
# Convert speed to meters per minute, percent grade to a decimal,
# and weight to kilos.
mpm = 26.8 * mph
incline /= 100
kg = pounds / 2.2

# Now we are ready for the oxygen formula:  ml per kg per minute.
# It assumes 3.7 mph represents walking and uses different formula to
running.
if mph <= 3.7:
    oxygen = 0.1 * mpm + 1.8 * mpm * incline + 3.5
else:
    oxygen = 0.2 * mpm + 0.9 * mpm * incline + 3.5

# Now calculate calories per minute, and the total calories.
cpm = oxygen * kg / 200
calories = cpm * minutes


# Output.  Precede the output with \n to visually separate input from output.
print("\nYour speed was {0:.1f} meters per minute.".format(mpm))
print("You weigh {0:.1f} kg.".format(kg))
print("You consumed {0:.1f} ml of oxygen per kg per minute.".format(oxygen))
print("You burned {0:.1f} calories per minute.".format(cpm))
print("Finally, you burned {0:.1f} calories.".format(calories))
```

More practice exercises

8. In Albuquerque, bad weather is often heralded by what is called the East Canyon wind. According to www.theweatherprediction.com/weatherpapers/011/index.html, meteorologists have determined a formula for predicting an East Canyon wind. It goes like this: Take the pressure readings (in millibars) from Albuquerque, Colorado Springs and Amarillo. You should expect an East Canyon wind if the average of the pressures at Colorado Springs and Amarillo exceeds that at Albuquerque by at least 5. How would we encode this algorithm in Python?

9. Ask the user to enter the lengths of 3 sides of a triangle. Determine if this triangle is equilateral (all sides equal), or isosceles (two sides equal) or scalene (no sides equal).

10. Ask the user to enter the 3 sides of a triangle. Determine which side is the longest and call this side "c", and call the other 2 sides "a" and "b". Determine if it is a right triangle by testing if $c^2 = a^2 + b^2$.

11. Given a number of seconds (which could be a large number), output the equivalent time in minutes:seconds, or hours:minutes:seconds as appropriate. For example, if the input is 200, then we would say this is 3 minutes and 20 seconds.

12. Ask the user for an integer, and determine if it's a 4-digit number or not.

13. A bank would like to devise a scheme for assigning account numbers so that one of the digits, say the last digit, is a "check digit." The purpose is to avoid referencing an invalid account number. The check digit should be computable from the rest of the digits of the number. And it should not be as trivial as saying "all account numbers must end in 3." Suggest a scheme for computing the check digit, and write an algorithm that will calculate it when given an account number.

14. Given a certain number of cents (1-99), tell the user how to make change. In other words, we want to know how many quarters, dimes, nickels and pennies are needed to make this amount of money. If a certain denomination is not needed, skip it in the output. For example, for 80 cents, only mention the quarters and nickels.

15. Show how we can use Python to implement the following tax table, which applies to single taxpayers on the 2018 South Carolina state income tax, according to taxfoundation.org/state-individual-income-tax-rates-brackets-2018/

If income is at least	but less than	Tax is	Plus this percent	Of the amount over
2970	5940	0	3	2970
5940	8910	89	4	5940
8910	11880	208	5	8910
11880	14860	357	6	11880
14860	---	535	7	14860