Loops

Doing Repetition in Python

If you wanted to repeat the dance behavior 10 times, all you have to do is:

```
for i in range(10):
   dance()
```

Repeat for a given amount; a count.

This is a new statement in Python: the `for`-statement. It is also called a *loop statement* or simply a *loop*. It is a way of repeating something a fixed number of times. The basic syntax of a `for`-loop in Python is:

This translates to: Do the set of commands below for 10 times.

```
for <variable> in <sequence>:
   <do something>
   <do something>
   ...
```

Remember: The flow of execution flows from the top down and never changes.

In Calico, this must be a range (0, x) where x is the desired number of repetitions.

The loop specification begins with the keyword `for` which is followed by a `<variable>` and then the keyword `in` and a `<sequence>` followed by a colon (`:`). This line sets up the number of times the repetition will be repeated. What follows is a set of statements, indented (again, indentation is important), that are called a *block* that forms the *body* of the loop (stuff that is repeated).

When executed, the `<variable>` (which is called a *loop index variable*) is assigned successive values in the <sequence> and for each of those values, the statements in the body of the loop are executed. A `<sequence>` in Python is a list of values. *Lists* are central to Python and we will see several examples of lists later. For now, look at the dance example above and notice that we have used the function `range(10)` to specify the sequence. To see what this function does you can start IDLE and enter it as an expression:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The result of entering the `range(10)` is a sequence (a list) of ten numbers 0..9. Notice how range returns a sequence of values starting from 0 all the way up to, but including, 10. Thus, the variable `i` in the loop:

```
for i in range(10):
   dance()
```

will take on the values 0 through 9 and for each of those values it will execute the `dance()` command.

Do This: Let us try this out on the robot. Modify the robot program from the start of this chapter to include the dance function and then write a main program to use the loop above.

```
# File: dance.py
# Purpose: A simple dance routine

# First import myro and connect to the robot

from myro import *
init()

# Define the new functions...

def yoyo(speed, waitTime):
        forward(speed, waitTime)
        backward(speed, waitTime)
        stop()

def wiggle(speed, waitTime):
        motors(-speed, speed)
        wait(waitTime)
        motors(speed, -speed)
        wait(waitTime)
        stop()

def dance():
        yoyo(0.5, 0.5)
        yoyo(0.5, 0.5)
        wiggle(0.5, 1)
        wiggle(0.5, 1)

# The main dance program
def main():
        print "Running the dance routine..."

        for danceStep in range(10):
            dance()

        print "...Done"

main()
```

For convenience:

You should save all of your definitions into a "start.py" file so that they will all compile at once.

To compile your "start.py":

from start import *

(just like when you import MYRO.

Notice that we have used `danceStep` (a more meaningful name than `i`) to represent the loop index variable. When you run this program, the robot should perform the dance routine ten times. Modify the value specified in the `range` command to try out some more steps. If you end up specifying a really large value, remember that for each value of `danceStep` the robot will do something

for 6 seconds. Thus, if you specified 100 repetitions, the robot will run for 10 minutes.

To repeat an action for a specific amount of time, use the following:

```
for seconds in timer(<seconds>):
    <do something>
    <do something>
```

This is essentially a modification of the previous 'for' loop, but the variable=seconds and the range=timer().

In writing robot programs there will also be times when you just want the robot to keep doing its behaviors forever! While technically by *forever* we do mean eternity in reality what is likely to happen is either it runs out of batteries, or you decide to stop it (by hitting CTRL-C). The Python command to specify this uses a different loop statement, called a `while`-loop that can be written as:

```
while True:
    <do something>
    <do something>
    ...
```

`True` is also a value in Python (along with `False`) about which we will learn more a little later. For now, it would suffice for us to say that the above loop is specifying that the body of the loop be executed forever!

Do This: Modify the `dance.py` program to use each of the `while`-loops instead of the `for`-loop. In the last case (`while TRUE:`) remember to use CTRL-C to stop the repetitions (and the robot).

As we mentioned above, repetition is one of the key concepts in computing. For example, we can use repetition to predict the world population in ten years by repeatedly computing the values for each year:

```
for year in range(10):
    population = population * (1 + growthRate)
```

That is, repeatedly add the increase in population, based on growth rate, ten times.

Do This: Modify the `worldPop.py` program to input the current population, growth rate, and the number of years to project ahead and compute the resulting total population. Run your program on several different values (Google: "world population growth" to get latest numbers). Can you estimate when the world population will become 9 billion?

Summary

This chapter introduced the basic structure of Python (and robot) programs. We also learned about *names* and *values* in Python. Names can be used to designate functions and values. The latter are also called *variables*. Python provides several different types of values: integers, floating point numbers, strings, and also boolean values (`True` and `False`). Most values have built-in operations (like addition, subtration, etc.) that perform calculations on them. Also, one can form sequences of values using lists. Python provides simple built-in facilities for obtaining input from the user. All of these enable us to write not only robot programs but also programs that perform any kind of computation. Repetition is a central and perhaps the most useful concept in computing. In Python you can specify repetition using either a `for`-loop or a `while`-loop. The latter are useful in writing general robot brain programs. In later chapters, we will learn how to write more sophisticated robot behaviors.

Myro Review

The computer converts the text in <something> and speaks it out loud. <something> is also simultaneously printed on the screen. Speech generation is done asynchronously. That is, anything following the speak command is done immediately after the entire string is spoken.

The computer converts the text in <something> to speech and speaks it out loud. <something> is also simultaneously printed on the screen. Speech generation is done asynchronously. That is, execution of subsequent commands can be done immediately after the entire string is spoken.

`timeRemaining(<seconds>)`

This is a way to specify timed repetitions in a while-loop (see below).

Python Review

Values

Values in Python can be numbers (integers or floating point numbers) or strings. Each type of value can be used in an expression by itself or using a combination of operations defined for that type (for example, +, -, *, /, % for numbers). Strings are considered sequences of characters (or letters).

Names

A name in Python must begin with either an alphabetic letter (a-z or A-Z) or the underscore (i.e. _) and can be followed by any sequence of letters, digits, or underscore letters.

```
input(<prompt string>)
```

This function prints out `<prompt string>` in the IDLE window and waits for the user to enter a Python expression. The expression is evaluated and its result is returned as a value of the input function.

```
from myro import *
initialize("comX")

<any other imports>
<function definitions>
def main():
    <do something>
    <do something>
    ...

main()
```

This is the basic structure of a robot control program in Python. Without the first two lines, it is the basic structure of all Python programs.

```
print <expression1>, <expression2>, ...
```

Prints out the result of all the expressions on the screen (in the IDLE window). Zero or more expressions can be specified. When no expression is specified, it prints out an empty line.

```
<variable name> = <expression>
```

This is how Python assigns values to variables. The value generated by <expression> will become the new value of <variable name>.

`range(10)`
Generates a sequence, a list, of numbers from 0..9. There are other, more general, versions of this function. These are shown below.

`range(n1, n2)`
Generates a list of numbers starting from n1…(n2-1). For example, range(5, 10) will generate the list of numbers [5, 6, 7, 8, 9].

`range(n1, n2, step)`
Generates a list of numbers starting from n1…(n2-1) in steps of step. For example, range(5, 10, 2) will generate the list of numbers [5, 7, 9].

Repetition

Loops

```
for <variable> in <sequence>:
    <do something>
    <do something>
    ...
```

must be a range value: (0,2) = repeat twice

```
for seconds in timer(<seconds>):
    <do something>
    <do something
```

```
while True:
    <do something>
    <do something>
    ...
```

These are different ways of doing repetition in Python. The first version will assign `<variable>` successive values in `<sequence>` and carry out the body once for each such value. The second version will carry out the body for `<seconds>` amount of time. `timeRemaining` is a Myro function (see above). The last version specifies an un-ending repetition.

Exercises

1. Write a Python program to convert a temperature from degrees Celsius to degrees Fahrenheit. Here is a sample interaction with such a program:

```
Enter a temperature in degrees Celsius: 5.0
That is equivalent to 41.0 degrees Fahrenheit.
```

Skip for now.

The formula to convert a temperature from Celsius to Fahrenheit is: `C/5=(F-32)/9`, where `C` is the temperature in degrees Celsius and `F` is the temperature in degrees Fahrenheit.

2. Write a Python program to convert a temperature from degrees Fahrenheit to degrees Celsius.

3. Write a program to convert a given amount of money in US dollars to an equivalent amount in Euros. Look up the current exchange rate on the web (see xe.com, for example).

4. Modify the version of the dance program above that uses a `for`-loop to use the following loop:

```
for danceStep in [1,2,3]:
    dance()
```

That is, you can actually use a list itself to specify the repetition (and successive values the loop variable will take). Try it again with the lists `[3, 2, 6]`, or `[3.5, 7.2, 1.45]`, or `["United", "States", "of", "America"]`. Also try replacing the list above with the string `"ABC"`. Remember, strings are also sequences in Python. We will learn more about lists later.

5. Run the world population program (any version from the chapter) and when it prompts for input, try entering the following and observe the behavior of the program. Also, given what you have learned in this chapter, try and explain the resulting behavior.

6. Use the values 9000000000, and 1.42 as input values as above. Except, when it asks for various values, enter them in any order. What happens?

7. Using the same values as above, instead of entering the value, say 9000000000, enter 6000000000+3000000000, or 450000000*2, etc. Do you notice any differences in output?

8. For any of the values to be input, replace them with a string. For instance enter `"Al Gore"` when it prompts you for a number. What happens?

9. Rewrite your solution to Exercise 4 from the previous chapter to use the program structure described above.

10. You were introduced to the rules of naming in Python. You may have noticed that we have made extensive use of *mixed case* in naming some entities. For example, `waitTime`. There are several naming conventions used by programmers and that has led to an interesting culture in of itself. Look up the phrase *CamelCase controversy* in your favorite search engine to learn about naming conventions. For an interesting article on this, see The Semicolon Wars (www.americanscientist.org/issues/pub/the-semicolon-wars).

11. Experiment with the `speak` function introduced in this chapter. Try giving it a number to speak (try both integers and floating point numbers). What is the largest integer value that it can speak? What happens when this limit is exceeded? Try to give the `speak` function a list of numbers, or strings, or both.

~~**12.** Write a Python program that sings the ABC song. *ABCD…XYZ. Now I know my ABC's. Next time won't you sing with me?*~~

Our Scribblers will not actually "speak". Instead, the "speak" function works more as a "print" function. Instead of hearing the words spoke n by the Scribbler, they will appear as written text in the Calico output window.

Extra Credit Assignment:
Make a Prezi presentation (www.prezi.com) on what you have learned about naming conventions from it he research prompt in this question. Your presentation must include works cited in order to earn the extra credit..