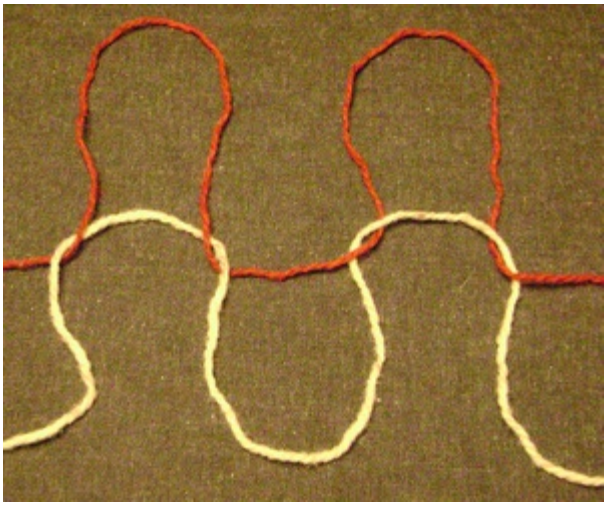
PLTW COMPUTER SCIENCE

Activity 1.3.7

For Loops

Introduction

As you've learned in the previous activities, computers can calculate and make decisions. A single calculation or decision would be unimpressive. Computers (and brains!) are impressive because they can make billions of calculations and decisions per second. Most programs don't have billions of instructions. A small handful of instructions repeated in a loop can be very powerful. In *Python*®, for and while loops are two of the control structures for iteration.



Iteration is a powerful idea even without computers. In knitting for example, a simple pair of stitches (knit and purl shown above) can be repeated with iteration in various patterns. What is something you enjoy doing that relies on iteration?

Materials

- Computer with Enthought Canopy distribution of *Python*® programming language

Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills and establish norms.

2. Launch Canopy and open an editor window.

3. If your teacher directs you to turn in your work with an IPython log, set the working directory for the IPython session and turn on session logging.

```
In []: %logstart -ort JDoeBSmith1_3_7.log
In []: # Jane Doe 1.3.7 IPython log
```

4. Start a new program in the code editor by choosing **File > New > Python file**. Save the file as `JDoe_BSmith_1_3_7.py`.

Part I: `for` loops, `range()`, and `help()`

5. You can loop over a block of code once for each item in a list, a tuple, or any other iterable data structure. Here we show a `for` loop using the list `numbers`, but you can use any iterable. You can make up any variable name for `item` here, and it will be assigned to each element in turn for each iteration through the loop:

```
In []: numbers = [1, 2, 3, 4]
In []: for item in numbers:
...:       print(item**2)
...:
1
4
9
16
```

The colon is required, indicating code will follow. The indentation tells the *Python* interpreter which block of code to repeat. Just as the Scratch™ programming language grouped the code to be repeated as a chunk inside a looping block, *Python* uses indentation to group code. Always use 4 spaces for each level of indentation.

You might have noticed that the output didn't come after an `Out[]:`. Recall that this is because `print` sends output to the screen but does not return a value.

6. A handy list for making loops is returned by the function:

```
range([start,] stop [, step])
```

This is a reference description of this function. Within this kind of notation, square brackets are used to mean "this part is optional." Italics are used to mean "this word should be replaced with a value for what the word describes." You can find this kind of information by using the

help function:

```
In []: help(range)
```

This built-in documentation is one place for reference, but there are many sources for additional help in programming. The official *Python* site has both tutorial help and reference material. Note that we are using *Python* version 2.7 because of the powerful libraries available:

http://docs.python.org/2.7

The reference material on `range()` shows the **function name** `range` and the parentheses that are there for every function. Inside the parentheses are the **arguments**, separated by commas. In the case of `range()`, there are three arguments: they are listed as `start, stop, step`. The square brackets around two of the arguments indicate that they are optional.

The `range()` function will return a list that begins at `start` and keeps adding `step`, reporting results that do not go beyond `stop`. The argument `start` has a **default value** of 0, and `step` has a default value of 1.

```
In  []: range(4)
Out []: [0, 1, 2, 3]
In  []: range(20, 12, -3)
Out []: [20, 17, 14]
```

Write code with `range()` that will return the list shown here:

```
In []: # 6. range()
In []: # use range to get the output show below
Out[]: [4, 6, 8, 10]
```

7. Paste the following code at the bottom of your *Python* file in the code editor and execute it. Call the function from the IPython session. Explain the output using a multi-line comment next to the function in the *Python* file.

```
def days():
    ''' Explain the function here
    '''
    for day in 'MTWRFSS':
        print(day + 'day')
    for day in range(5,8):
        print('It is the ' + str(day) + 'th of September')

In []: days()
```

8. Once you import a package (e.g., `import random`), the IPython session keeps the package's functions in the **namespace**. However, each time you run a program from the code editor, Canopy interprets the program in a "clean" namespace that contains only the built-in functions. There are only a few built-in functions but they do include, for example, `range`(). Any packages you want to use in a program must be imported in the program. Do a quick Internet search to see if you can determine the names of some other built-in *Python* functions. List five additional *Python* functions below.

9. Try this code. Don't forget to call it from the IPython session.

```
import matplotlib.pyplot as plt # standard short name
import random

plt.ion() # sets "interactive on": figures redrawn when updated

def picks():
    a = [] # make an empty list

    # Why all the brackets below?
    # a += [  brackets here to add an iterable onto a list      ]
    #    random.choice(   [brackets here to choose from a list] )
    a += [random.choice([1, 3, 10])]

    for choices in range(5):
        a += [random.choice([1, 3, 10])]

    plt.hist(a)
    plt.show()
```