Turtle programming lab

This sheet contains background and suggested tasks for familiarizing yourself with Python. Specifically, we’ll be working with the turtles in JES.

Begin by starting JES. Start in the bottom panel, where you can ask it to print the results of some expressions:

```python
print "Hello there"
print "Hello" + " there"
print 1 + 2
print 2 * 3
print 3 - 4
print 4 / 2
print 4 / 3
```

These illustrate common operations on strings and integers. Strings are groups of characters with quotes around them. They can be combined with the + operator. Integers support common arithmetic operations, with * playing the role of \( \times \). One aspect of integer arithmetic that you might not expect is that division rounds down, as shown in the last example. If you want an answer that is not rounded down, one of the arguments needs to have a decimal point. Such numbers are called floating point numbers; try some arithmetic operations (including division) with them.

Sometimes we’ll want to store the results of an expression for later use. To do this, we can set a variable with code like the following:

```
x = 1 + 2
```

The computer will compute the value of the right hand side (3) and store this value into the variable \( x \). Variables can be any single word and capitalization matters so that \( x \) and \( X \) are different variables. The purpose of using them is to give names to values that we’ve previously computed. Once a value is stored in the variable \( x \), you can use it in expressions:

```
print x
print x + 3
x = x + 1
```

The last is an interesting case because \( x \) appears on both sides of the equal sign. This would not be allowed in algebra since subtracting \( x \) from both sides leaves \( 0 = 1 \), but the computer is happy with this. It simply computes the value of the right hand side and stores the result back into \( x \).

The next concept to learn is functions. These are sets of instructions that we’ve given a name. You enter them into the top panel. Here is an example:

```
def printDouble(x):
    print x*2
```

This is defining a function called \texttt{printDouble} that takes a single argument called \( x \). It consists of a single action, printing twice \( x \). Enter this into the top panel and hit “Load Program”. (You’ll be asked if you want to save the program (yes) and then to pick where to put it (somewhere in your home directory).) Then you can call the function with \texttt{printDouble(4)}, which will print 8. When the function is called, the argument that you use (4 in this case) is assigned to \( x \) and then each statement of the function is run.
Look at the code from yesterday’s class on the course webpage (http://courses.knox.edu/cs127; select the course calendar and then the code linked from yesterday) and enter the makeChange function. (You’ll need to hit “Load Program” again.) This function determines how to make a given amount of change by keeping the amount of money it still needs to return in the variable amount. Run this function on a couple of values and work through its code with at least one so that you understand how it works.

Finally, after all that background, we’re ready to start using turtles. Begin by creating a “world” for your turtles:

```
w = makeWorld()
```

(You can use a different name if you want, but I just called it `w`.) The capitalization of “makeWorld” must be exactly as I gave it, though; remember that the computer is sensitive to capitalization differences. This will pop up a window. Then create a “turtle” to move around in this world:

```
t = makeTurtle(w)
```

Again, you can use a different name for your turtle, but you need to capitalize as I did and tell the new turtle which world it lives in.

Now you’ll have a little turtle in your window. (You can make multiple turtles in the same world; each will appear at the same place, but in a different color.) You can have it move with commands like:

```
forward(t)
turnRight(t)
forward(t, 50)
```

This will cause the turtle to move, drawing a line as it goes. The second argument to forward tells the turtle how far to go; if you omit it, the turtle moves 100 units. The turtle will always draw as it moves unless you tell it to lift the pen with penUp. You can see other turtle commands by selecting them from the “Turtles” submenu of the “JES Functions” menu.

Just as with other commands, you can put turtle commands in functions. Copy the drawSquare function from yesterday’s class notes. After you load the program again, you’ll be able to call drawSquare with the name of your turtle as an argument.

At this point, you’re ready to write turtle functions that draw interesting shapes. I suggest beginning by modifying drawSquare so that it also takes the side length (so drawSquare(t, 50) would draw a square with side length 50). Remember to reload the program each time you edit your functions.