1 Scope

The scope of a variable is the area of the program that can access it. If a variable has global scope, it can be accessed from any part of the program. If a variable has local scope, it can only be accessed from certain parts (or locales) of the program.

Variables defined inside a function have a local scope that is limited to the body of the function. This program will crash because variable radius is not recognized outside the scope of the function body.

```python
def circle_area(diameter):
    radius = diameter / 2.0
    return 3.14159 * radius ** 2

circle_area(20)
print radius # causes NameError: name 'radius' is not defined
```

A function’s parameters also have a local scope. If instead of printing radius, we attempt to print diameter, we would get a similar NameError.

2 Frames and the frame stack

Given that variables have local scope, consider this example.

```python
def square(x):
    sqrd = x * x
    return sqrd

x = 4
result = square(x + 1)
```

There are in fact two distinct variables called x. In the main program, the variable x refers to 4. Inside the function definition, x is a parameter. When the function is called, the parameter refers to 5 (because when the function is called, the number 5 is passed in as the argument). So does x refer to 4 or 5 or both? How does python make sense of this?

When executing program, python keeps track of information in things called frames. A frame keeps track of the mapping between variable names and the data values to which they refer. The global frame keeps track of all function names and variables defined in “main” program (outside of any function). When a function is called, a new frame – named a call frame – is created. The new frame is stacked on top of existing frames, forming a stack. The new call frame keeps track of variables created within the function, including parameters. (Notice the connection with scope: variables with the same scope are placed together on the same frame.) When the function finishes and returns to its caller, the call frame is destroyed. (Again, a scope connection: after the function call, the function’s local variables are invisible because the frame that held them is gone.)
The python visualizer (see earlier handout) is a great tool for understanding the call stack. Notice that at this moment, there are two variables named \texttt{x}, one refers to 4 and the other refers to 5.

![Visualizer Diagram]

### 3 Can functions modify their arguments?

What does this program print?

```python
def abs_value(x):
    if x < 0:
        x = -x
    return x

x = -100
print "abs_value(x) =", abs_value(x)
print "x =", x
```

The function returns 100, so it prints \texttt{abs_value(x) = 100}. However, the print statement after the function prints \texttt{x = -100}. Huh? The assignment statement inside the function does change what variable \texttt{x} refers to. However, there are two \texttt{x} variables in this program! (Try it in the visualizer.)

The assignment statement is affecting only the \texttt{x} variable whose scope is local to the function.

General rule: if an \textit{immutable} object (such as an \texttt{int, float, str}) is passed in as argument, the function \textit{cannot} change the object. Assignment statements inside the function only affect \textit{local} variables. However, if a \textit{mutable} object is passed in as argument, function may modify it and you may see an effect after the function call is complete. Example: a function that takes an Image object and changes pixel colors... the image will be changed even after the function call completes.

---

**Summary of Key Points**

- Variables defined inside a function will not be recognized outside function.
- The same variable name can appear both in- and outside a function. The two variables may refer to different objects! Python uses frames to keep track of variables and objects.
- A variable on the outside and a parameter on the inside can refer to the same object. Immutable objects cannot be changed (assignment only changes which object a variable refers to); but mutable objects can be changed! We will see several examples of this later.